librecours.net

Conception des bases de données II : Relationnel

bdd2.pdf

utc Formation

Stéphane Crozat

Publiée le 16 février 2020

Table des matières

Algèbre relationnelle

I

A. Cours

Le modèle relationnel, et en particulier l'algèbre relationnel qui lui est associée, est aussi le fondement théorique du langage standard SQL, qui est utilisé pour manipuler les données stockées dans une BD.

1. Opérateurs fondamentaux : projection, restriction et jointure

Objectifs

Connaître et savoir utiliser les opérateurs relationnels de projection, restriction, produit et jointure.

a) Introduction

La représentation d'information sous forme relationnelle est intéressante car les fondements mathématiques du relationnel, outre qu'ils permettent une modélisation logique simple et puissante, fournissent également un ensemble de concepts pour manipuler formellement l'information ainsi modélisée.

Ainsi une algèbre relationnelle, sous forme d'un ensemble d'opérations formelles, permet d'exprimer des questions, ou requêtes, posées à une représentation relationnelle, sous forme d'expressions algébriques.

L'algèbre relationnelle est principalement composée par les cinq opérateurs de base et les trois opérateurs additionnels suivants :

- **Opérateurs de base**
  - Union
  - Différence
  - Projection
  - Restriction

- Produit cartésien

- **Opérateurs additionels**
  - Intersection
  - Jointure
  - Division

Fondamental : Algèbre relationnelle et SQL

Les questions formulées en algèbre relationnelle sont la base des questions formulées en SQL pour interroger une base de données relationnelle.

b) Employés et départements

[30 minutes]

Soit les deux relations EMP et DEPT ci-après.

```
1  EMP (#ENO, ENOM, PROF, SAL, COMM, DNO=>DEPT(DNO))
2  DEPT (#DNO, DNOM, DIR=>EMP(ENO), VILLE)
```

- ENO : numéro d'employé, clé
- ENOM : nom de l'employé
- PROF : profession (directeur n'est pas une profession)
- SAL : salaire
- COMM : commission (un employé peut ne pas avoir de commission)
- DNO : numéro de département auquel appartient l'employé
- DNO : numéro de département, clé
- DNOM : nom du département
- DIR : numéro d'employé du directeur du département
- VILLE : lieu du département (ville)

Écrire en algèbre relationnelle les requêtes permettant d'obtenir les informations suivantes.

Question 1

Lister les employés ayant des revenus supérieurs à 10.000 euros.

Question 2

Trouver le nom et la profession de l'employé numéro 10.

Question 3

Lister les noms des employés qui travaillent à Paris.

Question 4

Trouver le nom du directeur du département *Commercial*.

Question 5

Trouver les professions des directeurs des départements.

Question 6

Trouver le nom des directeurs de département ayant comme profession *Ingénieur*.

c) Projection

Définition : Projection

La projection est une opération unaire (c'est à dire portant sur une seule relation).

La projection de R1 sur une partie de ses attributs {A1, A2, ...} produit une relation R2 dont le schéma est restreint aux attributs mentionnés en opérande, comportant les mêmes tuples que R1, et dont les doublons sont éliminés.

Syntaxe

```
R = Projection (R1, a1, a2, ...)
```

Exemple

Soit la relation suivante : `Personne (nom, prénom, age)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Tableau 1 Personne

Soit l'opération : `R = Projection (Personne, nom, age)`

On obtient alors la relation R composée des tuples suivants :

nom	age
Dupont	20
Durand	30

Tableau 2 R

Remarque : La projection élimine les doublons

Après suppression d'une partie des attributs du schéma, la relation peut comporter des doublons. Étant donné que l'on ne pourrait plus identifier ces doublons les uns par rapport aux autres, la seule solution sensée est donc de considérer que deux doublons sont équivalents, et donc de n'en garder qu'un seul dans la relation résultante.

Complément : Syntaxes alternatives

```
R = π (R1, a1, a2, ...)
```

$R = \pi_{a1,a2,...}(R1)$

d) Restriction

Définition : Restriction

La restriction est une opération unaire (c'est à dire portant sur une seule relation). La restriction de R1, étant donnée une condition C, produit une relation R2 de même schéma que R1 et dont les tuples sont les tuples de R1 vérifiant la condition C.

Syntaxe

```
R = Restriction (R1, condition)
```

Exemple

Soit la relation suivante : `Personne (nom, prénom, age)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Tableau 3 Personne

Soit l'opération suivante : `R = Restriction (Personne, age>25)`

On obtient alors la relation R composée de l'unique tuple restant suivant :

nom	prénom	age
Durand	Jean	30

Tableau 4 R

Complément : Syntaxes alternatives

`R = σ (R1, condition)`

$R = \sigma_{condition}(R1)$

Complément : Sélection

On utilise parfois sélection comme synonyme de restriction, mais il vaut mieux ne pas utiliser ce terme qui prend un sens différent en SQL.

e) Produit

Définition : Produit cartésien

Le produit cartésien est une opération binaire (c'est à dire portant sur deux relations). Le produit de R1 par R2 (équivalent au produit de R2 par R1) produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble des combinaisons possibles entre les tuples de R1 et ceux de R2.

Synonymes : Produit

Syntaxe

`R = Produit (R1, R2)`

Exemple

Soit les deux relations suivantes : `Personne (nom, prénom, age)` et `Voiture (type, marque)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Tableau 5 Personne

type	marque
Tesla	Model X
Citroën	2 CV

Tableau 6 Voiture

Soit l'opération suivante : `R = Produit (Homme, Voiture)`
On obtient alors la relation R composée des tuples suivants :

nom	prénom	age	type	marque
Dupont	Pierre	20	Tesla	Model X
Dupont	Pierre	20	Citroën	2 CV
Durand	Jean	30	Tesla	Model X
Durand	Jean	30	Citroën	2 CV

Tableau 7 R

Remarque

Le produit cartésien est rarement utilisé seul, mais il est à la base de la jointure.

Remarque

- Le nombre de tuples résultant du produit de R1 par R2 est égal au nombre de tuples de R1 **fois** le nombre de tuples de R2.
- Le nombre de colonne du produit de R1 par R2 est égal au nombre de colonne de R1 **plus** le nombre de colonnes de R2.

Complément : Syntaxes alternatives

```
R = X (R1, R2)
R = R1 X R2
```

f) Jointure

Définition : Jointure

La jointure est une opération binaire (c'est à dire portant sur deux relations). La jointure de R1 et R2, étant donné une condition C portant sur des attributs de R1 et de R2, **de même domaine**, produit une relation R3 ayant pour schéma la juxtaposition de ceux des relations R1 et R2 et pour tuples l'ensemble de ceux obtenus par concaténation des tuples de R1 et de R2, et qui vérifient la condition C.

Syntaxe

```
R = Jointure (R1, R2, condition)
```

Exemple

Soit les deux relations suivantes : `Personne (nom, prénom, age)` et `Voiture (type, marque, propriétaire)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Tableau 8 Personne

type	marque	propriétaire
Tesla	Model X	Dupont
Citroën	2 CV	Durand
Citroën	3 CV	Dupont

Tableau 9 Voiture

Soit l'opération suivante : `R = Jointure (Personne, Voiture, Personne.Nom=Voiture.Propriétaire)`

On obtient alors la relation R composée des tuples suivants :

nom	prénom	age	type	marque	propriétaire
Dupont	Pierre	20	Tesla	Model X	Dupont
Dupont	Pierre	20	Citroën	3 CV	Dupont
Durand	Jean	30	Citroën	2 CV	Durand

Tableau 10 R

Fondamental

La jointure est l'opération qui permet de rassembler les informations séparées entre plusieurs tables et référencées par des clés étrangères.

Remarque : Opération additionnelle

La jointure n'est pas une opération de base, elle peut être réécrite en combinant le produit et la restriction.

Complément : Équi-jointure

Une équi-jointure est une jointure dont la condition est un test d'égalité.

Complément : Syntaxes alternatives

`R = ⋈ (R1, R2, condition)`

`R = R1 ⋈(condition) R2`

g) Exercice

Quelles sont les expressions relationnelles équivalentes à :

```
1 Projection ( Jointure (R1, R2, R1.A1=R2.A1), R1.A1, R2.A2)
```

- [ ] Jointure (Projection(R1, A1), Projection(R2, A2), R1.A1=R2.A1)
- [ ] Projection (Jointure (R2, R1, R2.A1=R1.A1), R1.A1, R2.A2)
- [ ] Projection (Restriction (Produit(R1, R2), R1.A1=R2.A1), R1.A1, R2.A2)
- [ ] Produit (R1, R2, R1.A1=R2.A1, R1.A1, R2.A2)

2. Opérateurs complémentaires

Objectifs

Maîtriser l'algèbre relationnelle.

a) Jointure naturelle

Définition : Jointure naturelle

La jointure naturelle entre R1 et R2 est une jointure pour laquelle la condition est l'égalité entre les attributs de même nom de R1 et de R2. Il est donc inutile de spécifier la condition dans une jointure naturelle, elle reste toujours implicite.

Syntaxe

```
R = JointureNaturelle (R1, R2)
```

Exemple

Soit deux relations R1 (A, B, C) et R2 (A, D), l'opération `Jointure(R1,R2,R1.A=R2.A)` est équivalente à l'opération `JointureNaturelle(R1,R2)`.

Remarque

Pour appliquer une jointure naturelle, il faut que les deux relations opérandes aient au moins un attribut ayant le même nom en commun.

Complément : Syntaxes alternatives

```
R = ⋈ (R1, R2)
R = R1 ⋈ R2
```

b) Jointure externe

Introduction

La jointure est une opération qui entraîne la perte de certains tuples : ceux qui appartiennent à une des deux relations opérandes et qui n'ont pas de correspondance dans l'autre relation. Il est nécessaire dans certains cas de pallier cette lacune, et l'on introduit pour cela la notion de jointure externe.

Définition : Jointure externe

La jointure externe entre R1 et R2 est une jointure qui produit une relation R3 à laquelle on ajoute les tuples de R1 et de R2 exclus par la jointure, en complétant avec des valeurs nulles pour les attributs de l'autre relation.

Définition : Jointure externe gauche

La jointure externe gauche entre R1 et R2 est une jointure externe pour laquelle on ajoute seulement les tuples de R1 (c'est à dire la relation de gauche) ayant été exclus.

Synonymes : Jointure gauche

Définition : Jointure externe droite

La jointure externe droite entre R1 et R2 est une jointure externe pour laquelle on ajoute seulement les tuples de R2 (c'est à dire la relation de droite) ayant été exclus.

Bien entendu une jointure externe droite peut être réécrite par une jointure externe gauche (et réciproquement) en substituant les relations opérandes R1 et R2.

Synonymes : Jointure droite

Syntaxe

```
R = JointureExterne (R1, R2, condition)
R = JointureExterneGauche (R1, R2, condition)
R = JointureExterneDroite (R1, R2, condition)
```

Exemple

Soit les deux relations suivantes : `Personne (nom, prénom, age)` et `Voiture (type, marque, propriétaire)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30
Martin	Georges	40

Tableau 11 Personne

type	marque	propriétaire
Tesla	Model X	Dupont
Citroën	2 CV	Durand
Citroën	3 CV	*NULL*

Tableau 12 Voiture

Soit l'opération suivante : `R = JointureExterne (Homme, Voiture, Homme.Nom=Voiture.Propriétaire)`

On obtient alors la relation R composée des tuples suivants :

nom	prénom	age	type	marque	propriétaire
Dupont	Pierre	20	Tesla	Model X	Dupont
Durand	Jean	30	Citroën	2 CV	Durand

nom	prénom	age	type	marque	propriétaire
Martin	Georges	40	NULL	NULL	NULL
NULL	NULL	NULL	Citroën	3 CV	NULL

Tableau 13 R

Une jointure externe gauche n'aurait renvoyé que les trois premiers tuples et une jointure externe droite n'aurait renvoyé que les deux premiers et le dernier tuple.

Complément : Syntaxes alternatives

`R = +⋈+ (R1, R2, condition)` (jointure externe)

`R = +⋈ (R1, R2, condition)` (jointure externe gauche)

`R = ⋈+ (R1, R2, condition)` (jointure externe droite)

`R = R1 +⋈+(condition) R2`

`R = R1 +⋈(condition) R2`

`R = R1 ⋈+(condition) R2`

c) Opérateurs ensemblistes

Attention

Les opérateurs ensemblistes sont des relations binaires (c'est à dire entre deux relations) portant sur des relations **de même schéma**.

Définition : Union

L'union de deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples appartenant à R1 et/ou à R2.

Définition : Différence

La différence entre deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples de R1 n'appartenant pas à R2. Notons que la différence entre R1 et R2 n'est pas égale à la différence entre R2 et R1.

Définition : Intersection

L'intersection de deux relations R1 et R2 de même schéma produit une relation R3 de même schéma constituée de l'ensemble des tuples appartenant à la fois à R1 et à R2. Notons que l'intersection n'est pas une opération de base, car elle est équivalent à deux opérations de différence successives.

Syntaxe

```
R = Union (R1, R2)
R = Différence (R1, R2)
R = Intersection (R1, R2)
```

Attention

Les opérateurs ensemblistes éliminent les doublons.

Exemple

Soit les deux relations suivantes : `Homme (nom, prénom, age)` et `Femme (nom, prénom, age)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Tableau 14 Homme

nom	prénom	age
Martin	Isabelle	24
Blanc	Hélène	25

Tableau 15 Femme

Soit l'opération suivante : `R = Union (Homme, Femme)`

On obtient alors la relation R composée des tuples suivants :

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30
Martin	Isabelle	24
Blanc	Hélène	25

Tableau 16 R

La différence entre Homme et Femme (respectivement entre Femme et Homme) renvoie la relation Homme (respectivement Femme), car aucun tuple n'est commun aux deux relations. L'intersection entre Homme est Femme est vide, pour la même raison.

Complément : Union externe

Il est possible de définir une opération d'union externe, qui permet de réaliser l'union de deux relations de schéma différent, en ramenant les relations aux mêmes schémas, et en les complétant avec des valeurs nulles.

Complément : Syntaxes alternatives

`R = R1 ∪ R2`

`R = R1 ∩ R1`

`R = R1 − R2`

d) Division

Définition : Division

La division est une opération binaire (c'est à dire portant sur deux relations).

La division de R_D par R_d a pour résultat R_Q tel que R_Q comporte le plus grand

ensemble possible de tuples qui concaténés à ceux de R_d donnent toujours un tuple de R_D.

Fondamental

$R_Q = R_D \div R_d$ si et seulement si :

- **R_d X $R_Q \subseteq$ dans R_D**
- $\nexists R_{Q'}$ tel que $R_Q \subset R_{Q'}$ et R_d X $R_{Q'} \subseteq$ dans R_D

Syntaxe

`R_Q = Division (R_D, R_d)`

Attention

- Tous les attributs de R_d sont des attributs de R_D (c'est à dire de même nom et de même domaine), sinon ∀R, `R_d X R` donnera des tuples dont au moins un attribut n'appartient pas au schéma de R_D.
- R_D a au moins un attribut de plus que R_d, sinon ∀R, `R_d X R` donnera des tuples qui ont au moins un attributs de plus que R_D.
- R_Q comporte les attributs appartenant à R_D mais n'appartenant pas à R_d

Conseil : Comparaison avec la division entière

- R_D est le dividende
- R_d est le diviseur
- R_Q est le quotien (tel que R_Q x R_d donne le plus grand entier inférieur à R_D)
- Il y a une sorte de reste R_r qui contient les tuples de R_D qui ne sont pas retrouvés par l'opération R_d X R_Q.

Exemple

Soit les deux relations suivantes : `(R_D) Pratique (personne, age, métier)` et `(R_d) Métier (métier)`

personne	age	métier
Dupont	20	Ingénieur
Dupont	20	Professeur
Durand	30	Professeur
Martin	40	Ingénieur
Martin	40	Professeur

Tableau 17 (R_D) Pratique

métier
Ingénieur
Professeur

Tableau 18 (R_d) Métier

Soit l'opération suivante : `R_Q = Division (Pratique, Métier)`

On obtient alors la relation R_Q composée des tuples suivants :

personne	age
Dupont	20
Martin	40

Tableau 19 (R_Q) Résultat

On peut considérer que le tuple (Durand, 30, Professeur) est ici une sorte de reste, il n'est pas retrouvé par l'opération R_d X R_Q.

Méthode : Réponse aux questions : Pour tous les ...

La division permet de répondre aux questions du type : "Donnez toutes les personnes qui pratiquent tous les métiers de la relation métier".

Remarque : Opération additionnelle

La division n'est pas une opération de base, elle peut être réécrite en combinant le produit, la restriction et la différence.

Syntaxe : Syntaxes alternatives

$R_Q = \div (R_D, R_d)$

$R_Q = R_D \div R_d$

e) Renommage

Définition : Division

Le renommage est une opération qui permet de redéfinir les noms des attributs d'une relation R.

Syntaxe

```
R = Renommage (R1, a1, a2, ...)
```

Exemple

Soit la relation suivante : `Personne (nom, prénom, age)`

nom	prénom	age
Dupont	Pierre	20
Durand	Jean	30

Tableau 20 Personne

Soit l'opération suivante : `R = Renommage (Personne, a, b, c))`

On obtient alors la relation R composée des tuples suivants :

a	b	c
Dupont	Pierre	20
Durand	Jean	30

Tableau 21 R

Syntaxe : Syntaxes alternatives

R = ρ (R1, a1, a2...)

R = $\rho_{a1, a2, ...}$(R1)

f) Proposition de notations

Introduction

Il existe plusieurs syntaxes pour écrire des opérations d'algèbre relationnelle, certaines inspirées de l'algèbre classiques, d'autres reposant sur des notations graphiques. Nous proposons une notation fonctionnelle qui a le mérite d'être facile à écrire et d'être lisible.

Syntaxe

```
1  R = Union (R1, R2)
2  R = Différence (R1, R2)
3  R = Intersection (R1, R2)
4  R = Projection (R1, a1, a2, ...)
5  R = Restriction (R1, condition)
6  R = Produit (R1, R2)
7  R = Jointure (R1, R2, condition)
8  R = JointureNaturelle (R1, R2)
9  R = JointureExterne (R1, R2, condition)
10 R = JointureGauche (R1, R2, condition)
11 R = JointureDroite (R1, R2, condition)
12 R = Division (R1, R2)
13 R = Renommage (R, a1, a2, ...)
```

Exemple : Notation décomposée

```
1  R' = Restriction(R1, A1=1 AND A2=2)
2  R = Projection (R', A3)
```

g) Exercice

Soit les deux relations R1 et R2 suivantes, définies en extension :

A	B
1	A
2	B
3	C

Tableau 22 R1

A
1
2

Tableau 23 R2

Combien de tuples renvoie l'opération relationnelle suivante ?

```
1 R3 = JointureNaturelle (Intersection (Projection(R1,A), R2), R2)
```

h) Opérateurs de base et additionnels

Réécrivez les opérateurs additionnels suivants, à partir d'opérateurs de base :

Question 1

Réécrivez Intersection à partir de Différence

Question 2

Réécrivez Jointure à partir de Produit et Restriction

B. Exercices

1. Faire du Cinéma

[30 minutes]

On considère les deux relations suivantes :

```
1 FILMS (titre:string, pays:string, année:integer(4),
  réalisateur:string, durée:integer(3))
2 ACTEURS (titre:string, acteur:string)
```

La relation **FILMS** donne pour chaque film, identifié par son titre, le pays, l'année de sortie, réalisateur et la durée.

La relation **ACTEURS** donne pour chaque film l'ensemble des principaux acteurs.

À l'aide de l'algèbre relationnelle, exprimer les requêtes ci-après.

Question 1

Lister les films français (titre, année, réalisateur).

Question 2

Donnez les années de sortie des films dans lesquels l'acteur Jean GABIN a joué.

Question 3

Trouver les acteurs qui ont tourné avec François Truffaut comme réalisateur.

Question 4

Lister les films dans lesquels le réalisateur est aussi acteur.

Question 5

Trouver tous les acteurs qui ont été partenaires de l'actrice Catherine Deneuve.

Question 6

Lister les réalisateurs n'ayant joué comme acteurs que dans des films qu'ils ne réalisaient pas eux-mêmes.

Question 7

Lister les réalisateurs ayant joué comme acteurs dans des films qu'ils ne réalisaient pas eux-mêmes.

Question 8

Donnez les acteurs qui jouent dans tous les films de François TRUFFAUT.

2. Le retour des écoliers

[45 minutes]

Soit le schéma relationnel suivant :

```
1  IMMEUBLE (#ADI, NBETAGES, DATEC, PROP)
2  APPIM (#ADI, #NAPR, OCCUP, TYPE, SUPER, ETAGE)
3  PERSONNE (#NOM, AGE, PROF, ADR, NAPR)
4  ÉCOLE (#NOMEC, ADEC, DIR)
5  CLASSE (#NOMEC, #NCL, MAITRE)
6  ENFANT (#NOMP, #PRENOM, AN, NOMEC, NCL)
```

Avec la signification suivante :

- **Relation IMMEUBLE**

  ADI : adresse d'immeuble, clé ; on fait l'hypothèse pour simplifier, que l'adresse identifie de manière unique un immeuble

  NBETAGES : nombre d'étages d'un immeuble

  DATEC : date de construction (année)

  PROP : nom du propriétaire de l'immeuble qui est une personne

- **Relation APPIM (Appartement)**

  ADI : adresse d'immeuble

  NAPR : numéro d'appartement

  OCCUP : occupant de l'appartement (nom de la personne ayant signé le contrat de location, éventuellement aucun)

  TYPE : type de l'appartement (Studio, F2, ...)

  SUPER : superficie de l'appartement

  ETAGE : étage où se situe l'appartement

- **Relation PERSONNE**

  NOM : nom de personne, clé ; on fait l'hypothèse pour simplifier, que ce nom est unique sur l'ensemble des personnes que l'on considère dans la base

  AGE : âge de la personne

  PROF : profession de la personne

  ADR : adresse de la résidence d'une personne, il s'agit d'un immeuble

  NAPR : numéro d'appartement

- **Relation ÉCOLE**

NOMEC : nom d'une école, clé

ADEC : adresse d'une école

DIR : nom du directeur

- **Relation CLASSE**

  NOMEC : nom d'une école

  NCL : nom de la classe, e.g., CP1, CE2, CE3, etc...

  MAITRE : nom de l'instituteur

- **Relation ENFANT**

  NOMP : nom de la personne responsable de l'enfant, clé e.g., père, mère etc...

  PRENOM : prénom de l'enfant

  AN : année de naissance

  NOMEC : nom d'une école

  NCL : nom de la classe

La relation IMMEUBLE décrit un ensemble d'immeubles. Chaque immeuble a un propriétaire. La relation APPIM décrit pour chaque immeuble l'ensemble des appartements qui le compose (il y a au mimimum un appartement par immeuble). Chaque appartement peut héberger plusieurs personnes mais il y en a une qui est responsable (par exemple la personne qui a signé le contrat de location) et qui est désignée par l'attribut OCCUP. Si l'appartement est inoccupé, il prend la valeur NULL. La relation PERSONNE décrit un ensemble de personnes. ADR et NAPR représentent l'adresse où réside une personne. Une personne peut avoir plusieurs enfants décrits par la relation ENFANT. Pour simplifier, on ne considère que les enfants allant à l'école primaire. Les écoles et les classes sont décrites dans les relations ÉCOLE et CLASSE, chaque école est composée au minimum d'une classe et chaque classe est au moins fréquentée par un enfant.

Question 1

Donner l'adresse des immeubles ayant plus de 10 étages et construits avant 1970.

Question 2

Donner les noms des personnes qui habitent dans un immeuble dont ils sont propriétaires.

Question 3

Donner les noms des personnes qui ne sont pas propriétaires.

Question 4

Donner les adresses des immeubles possédés par des informaticiens dont l'âge est inférieur à 40 ans .

Question 5

Donner la liste des occupants (nom, âge, profession) des immeubles possédés par DUPONT.

Question 6

Donner le nom et la profession des propriétaires d'immeubles dans lesquels il y a des appartements vides.

Question 7

Donner les noms des maîtres qui habitent dans le même immeuble (à la même adresse) qu'au moins un de leurs élèves (on suppose que les enfants vivent sous le même toit que leur parents).

Question 8

Donner l'adresse de l'immeuble, la date de construction, le type d'appartement et l'étage où habitent chacun des maîtres des enfants de DUPONT.

3. Quiz : Algèbre relationnelle

Exercice 1

Quelles sont les opérations relationnelles, qui appliquées sur les relations instanciées ci-dessous, renvoient un ensemble non nul de tuples ?

Num	Nom	Famille
1	Ours	Mammifère
2	Truite	Poisson
3	Homme	Mammifère
4	Martin-pêcheur	Oiseau

Tableau 24 Animal

Num	Nom
1	Forêt
2	Montagne
3	Ciel
4	Rivière
5	Mer

Tableau 25 Environnement

Animal	Environnement
1	1
1	2
1	4
2	4
4	3

Tableau 26 Habiter

Mangeur	Mangé	Fréquence
1	2	Souvent
1	3	Rarement
1	4	Rarement
4	2	Souvent
3	1	Rarement
3	2	Souvent

Tableau 27 Manger

- [ ] Restriction(Projection (Animal, Nom, Famille), Famille='Mammifère')
- [ ] Restriction(Jointure(Jointure (Animal, Habiter, Animal.Num=Habiter.Animal), Environnement, Environnement.Num=Habiter.Environnement), Animal.Num='3')
- [ ] Restriction(JointureExterneGauche(Animal, Habiter, Animal.Num=Habiter.Animal), Animal.Nom='Homme')
- [ ] Jointure(Animal, Manger, Animal.Num=Manger.Mangeur AND Animal.Num=Manger.Mangé)

Exercice 2

Soit le schéma relationnel :

```
1 R1(X, Y)
2 R2(X, Y)
```

Quelles sont les opérations relationnelles équivalentes à l'opération :

```
1 Projection( JointureNaturelle(R1,R2), R1.X)
```

- [ ] JointureNaturelle(Projection(R1, X), Projection(R2, X))
- [ ] Projection(Selection (Produit(R1, R2), R1.X=R2.X AND R1.Y=R2.Y), R1.X)
- [ ] Projection(Union(R1, R2), R1.X)
- [ ] Projection(JointureExterne (R1, R2, R1.X=R2.X AND R1.Y=R2.Y), R1.X)
- [ ] Projection(Jointure (R1, R2, R1.X=R2.X AND R1.Y=R2.Y), R1.X)

Exercice 3

Soit la relation instanciée suivante :

A	B	C
1	1	0
1	0	1
0	1	1

Tableau 28 Relation R1

Quelles relations sont retournées par l'opération relationnelle suivante :

```
1 R2 = JointureNaturelle(R1, R1)
```

1	1	0
1	0	1
0	1	1

Tableau 29 R2a

1	1	0
1	0	1
0	1	1
1	1	0
1	0	1
0	1	1

Tableau 30 R2b

1	1	1
0	0	0

Tableau 31 R2c

1	1	0	1	1	0
1	0	1	1	0	1
0	1	1	0	1	1

Tableau 32 R2d

- ☐ R2a
- ☐ R2b
- ☐ R2c
- ☐ R2d
- ☐ Une relation vide (aucun tuple)

II

Interrogation de bases de données SQL

A. Cours

SQL est un langage standardisé, implémenté par tous les SGBDR, qui permet, indépendamment de la plate-forme technologique et de façon déclarative, de définir le modèle de données, de le contrôler et enfin de le manipuler.

1. Questions simples avec le Langage de Manipulation de Données (SELECT)

a) Question (SELECT)

Fondamental : Question

La requête de **sélection** ou **question** est la base de la recherche de données en SQL.

Définition : Sélection

La sélection est la composition d'un **produit cartésien**, d'une **restriction** et d'une **projection** (ou encore la composition d'une jointure et d'une projection).

Syntaxe

```
1 SELECT liste d'attributs projetés
2 FROM liste de relations du produit cartésien
3 WHERE condition de la restriction
```

- La partie SELECT indique le sous-ensemble des attributs qui doivent apparaître dans la réponse (c'est le schéma de la relation résultat).

- La partie FROM décrit les relations qui sont utilisables dans la requête (c'est à dire l'ensemble des attributs que l'on peut utiliser).
- La partie WHERE exprime les conditions que doivent respecter les attributs d'un tuple pour pouvoir être dans la réponse. Une condition est un prédicat et par conséquent renvoie un booléen. Cette partie est optionnelle.

Exemple

```
1  SELECT Nom, Prenom
2  FROM Personne
3  WHERE Age>18
```

Cette requête sélectionne les attributs Nom et Prénom des tuples de la relation Personne, ayant un attribut Age supérieur à 18.

Syntaxe : Notation préfixée

Afin de décrire un attribut d'une relation en particulier (dans le cas d'une requête portant sur plusieurs relations notamment), on utilise la notation `relation.attribut`.

Exemple

```
1  SELECT Personne.Nom, Personne.Prenom, Vol.Depart
2  FROM Personne, Vol
3  WHERE Personne.Vol=Vol.Numero
```

*Syntaxe : SELECT **

Pour projeter l'ensemble des attributs d'une relation, on peut utiliser le caractère * à la place de la liste des attributs à projeter.

Exemple

```
1  SELECT *
2  FROM Avion
```

Cette requête sélectionne tous les attributs de la relation Avion.

Notons que dans cet exemple, la relation résultat est exactement la relation Avion

b) Opérateurs de comparaisons et opérateurs logiques

Introduction

La clause WHERE d'une instruction de sélection est définie par une condition. Une telle condition s'exprime à l'aide d'opérateurs de comparaison et d'opérateurs logiques. Le résultat d'une expression de condition est toujours un booléen.

Définition : Condition

```
1  Condition Elémentaire ::= Propriété <Opérateur de comparaison>
   Constante
2  Condition ::= Condition <Opérateur logique> Condition | Condition
   Elémentaire
```

Les opérateurs de comparaison sont :

- P = C
- P <> C
- P < C
- P > C
- P <= C
- P >= C
- P BETWEEN C1 AND C2
- P IN (C1, C2, ...)
- P LIKE 'chaîne'
- P IS NULL

Les opérateur logique sont :

- OR
- AND
- NOT

Remarque : Opérateur LIKE

L'opérateur `LIKE` 'chaîne' permet d'insérer des jokers dans l'opération de comparaison (alors que l'opérateur = teste une égalité stricte) :

- Le joker `%` désigne 0 ou plusieurs caractères quelconques
- Le joker _ désigne 1 et 1 seul caractère

On préférera l'opérateur = à l'opérateur `LIKE` lorsque la comparaison n'utilise pas de joker.

c) Renommage de colonnes et de tables avec les alias

Syntaxe : Alias de table

Il est possible de redéfinir le nom des relations au sein de la requête afin d'en simplifier la syntaxe.

```
1  SELECT t1.attribut1, t2.attribut2
2  FROM table1 t1, table2 t2
```

Exemple

```
1  SELECT Parent.Prenom, Enfant.Prenom
2  FROM Parent, Enfant
3  WHERE Enfant.Nom=Parent.Nom
```

Cette requête sélectionne les prénoms des enfants et des parents ayant le même nom. On remarque la notation Parent.Nom et Enfant.Nom pour distinguer les attributs Prenom des relations Parent et Enfant.

On notera que cette sélection effectue une jointure sur les propriétés Nom des relations Parent et Enfant.

Remarque : Alias d'attribut (AS)

Il est possible de redéfinir le nom des propriétés de la relation résultat.

```
1  SELECT attribut1 AS a1, attribut2 AS a2
```

```
2  FROM table
```

d) Dédoublonnage (SELECT DISTINCT)

Attention : SELECT DISTINCT

L'opérateur SELECT n'élimine pas les doublons (i.e. les tuples identiques dans la relation résultat) par défaut. Il faut pour cela utiliser l'opérateur SELECT DISTINCT.

Exemple

```
1  SELECT DISTINCT Avion
2  FROM Vol
3  WHERE Date=31-12-2000
```

Cette requête sélectionne l'attribut Avion de la relation Vol, concernant uniquement les vols du 31 décembre 2000 et renvoie les tuples sans doublons.

e) Tri (ORDER BY)

Introduction

On veut souvent que le résultat d'une requête soit trié en fonction des valeurs des propriétés des tuples de ce résultat.

Syntaxe : ORDER BY

```
1  SELECT liste d'attributs projetés
2  FROM liste de relations
3  WHERE condition
4  ORDER BY liste ordonnée d'attributs
```

Les tuples sont triés d'abord par le premier attribut spécifié dans la clause ORDER BY, puis en cas de doublons par le second, etc.

Remarque : Tri décroissant

Pour effectuer un tri décroissant on fait suivre l'attribut du mot clé "DESC".

Exemple

```
1  SELECT *
2  FROM Personne
3  ORDER BY Nom, Age DESC
```

f) Projection de constante

Syntaxe : Projection de constante

Il est possible de projeter directement des constantes (on utilisera généralement un alias d'attribut pour nommer la colonne).

```
1  SELECT constante AS nom
```

Cette requête renverra une table avec une seule ligne et une seule colonne à la valeur de *constante*.

Exemple

```
1  SELECT '1' AS num
```

```
1   num
2  -----
3   1
```

Exemple

```
1  SELECT 'Hello world' AS hw
```

```
1        hw
2  -------------
3   Hello world
```

Exemple

```
1  SELECT CURRENT_DATE AS now
```

```
1       now
2  ------------
3   2016-10-21
```

g) Commentaires en SQL

L'introduction de commentaires au sein d'une requête SQL se fait :

- En précédant les commentaires de `--` pour les commentaires mono-ligne
- En encadrant les commentaires entre `/*` et `*/` pour les commentaires multi-lignes.

Exemple

```
1  /*
2  * Creation of table Student
3  * Purpose : managing information about students in the system
4  */
5  CREATE TABLE person (
6  pknum VARCHAR(13) PRIMARY KEY, --Student national number
7  name VARCHAR(25)
8  );
9
10
```

h) Représentation de représentants

[30 minutes]

Soit le schéma relationnel et le code SQL suivants :

```
1  REPRESENTANTS (#NR, NOMR, VILLE)
2  PRODUITS (#NP, NOMP, COUL, PDS)
3  CLIENTS (#NC, NOMC, VILLE)
4  VENTES (#NR=>REPRESENTANTS(NR), #NP=>PRODUITS(NP), #NC=>CLIENTS(NC),
```

```
QT)
```

```
/* Les requêtes peuvent être testées dans un SGBDR, en créant une base de données avec le script SQL suivant */

/*
DROP TABLE VENTES ;
DROP TABLE CLIENTS ;
DROP TABLE PRODUITS ;
DROP TABLE REPRESENTANTS ;
*/

CREATE TABLE REPRESENTANTS (
 NR INTEGER PRIMARY KEY,
 NOMR VARCHAR,
 VILLE VARCHAR
);

CREATE TABLE PRODUITS (
 NP INTEGER PRIMARY KEY,
 NOMP VARCHAR,
 COUL VARCHAR,
 PDS INTEGER
);

CREATE TABLE CLIENTS (
 NC INTEGER PRIMARY KEY,
 NOMC VARCHAR,
 VILLE VARCHAR
);

CREATE TABLE VENTES (
 NR INTEGER REFERENCES REPRESENTANTS(NR),
 NP INTEGER REFERENCES PRODUITS(NP),
 NC INTEGER REFERENCES CLIENTS(NC),
 QT INTEGER,
 PRIMARY KEY (NR, NP, NC)
);

INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (1, 'Stephane', 'Lyon');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (2, 'Benjamin', 'Paris');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (3, 'Leonard', 'Lyon');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (4, 'Antoine', 'Brest');
INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (5, 'Bruno', 'Bayonne');

INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (1, 'Aspirateur', 'Rouge', 3546);
INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (2, 'Trottinette', 'Bleu', 1423);
INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (3, 'Chaise', 'Blanc', 3827);
INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (4, 'Tapis', 'Rouge', 1423);

INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (1, 'Alice', 'Lyon');
INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (2, 'Bruno', 'Lyon');
INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (3, 'Charles', 'Compiègne');
INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (4, 'Denis', 'Montpellier');
INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (5, 'Emile', 'Strasbourg');

INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 1, 1, 1);
```

```sql
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 1, 2, 1);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (2, 2, 3, 1);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (4, 3, 3, 200);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 2, 190);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 3, 2, 300);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 2, 120);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 4, 120);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 4, 2);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 1, 3);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 1, 5);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 3, 1);
INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 5, 1);
```

Écrire en SQL les requêtes permettant d'obtenir les informations ci-après.

Question 1

Tous les détails de tous les clients.

Question 2

Les numéros et les noms des produits de couleur rouge et de poids supérieur à 2000.

Question 3

Les représentants ayant vendu au moins un produit.

Question 4

Les noms des clients de Lyon ayant acheté un produit pour une quantité supérieure à 180.

Question 5

Les noms des représentants et des clients à qui ces représentants ont vendu un produit de couleur rouge pour une quantité supérieure à 100.

2. Opérations d'algèbre relationnelle en SQL

a) Expression d'une restriction

Syntaxe

```sql
SELECT *
FROM R
WHERE <condition>
```

Exemple

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	Delta	

R

A	B	C
1	Alpha	10
2	Bravo	10

```
Restriction(R1,C<20)

SELECT *
FROM R1
WHERE C<20
```

Tableau 33 Exemple de restriction (SQL et Algèbre)

b) Expression d'une projection

Syntaxe

```
1  SELECT P1, P2, Pi
2  FROM R
```

Exemple

R1	#A	B	C=>R2
	1	Alpha	10
	2	Bravo	10
	3	Charlie	20
	4	Delta	

R	A	C
	1	10
	2	10
	3	20
	4	

```
Projection(R1,A,C)

SELECT A, C
FROM R1
```

Tableau 34 Exemple de projection (SQL et Algèbre)

c) Expression du produit cartésien

Syntaxe

```
1  SELECT *
2  FROM R1, R2, Ri
```

Exemple

R1	#A	B	C=>R2
	1	Alpha	10
	2	Bravo	10
	3	Charlie	20
	4	Delta	

R2	#X	Y
	10	Echo
	20	Fox
	30	Golf

A	B	C	X	Y
1	Alpha	10	10	Echo
1	Alpha	10	20	Fox
1	Alpha	10	30	Golf
2	Bravo	10	10	Echo
2	Bravo	10	20	Fox
2	Bravo	10	30	Golf
3	Charlie	20	10	Echo
3	Charlie	20	20	Fox
3	Charlie	20	30	Golf
4	Delta		10	Echo
4	Delta		20	Fox
4	Delta		30	Golf

```
Produit(R1,R2)

SELECT *
FROM R1,R2
```

Tableau 35 Exemple de produit (SQL et Algèbre)

d) Expression d'une jointure

Syntaxe : Jointure par la clause WHERE

En tant que composition d'un produit cartésien et d'une restriction la jointure s'écrit :

```
1  SELECT *
2  FROM R1, R2, Ri
3  WHERE <condition>
```

Avec `Condition` permettant de joindre des attributs des `Ri`

Syntaxe : Jointure par la clause ON

On peut également utiliser la syntaxe dédiée suivante :

```
1  SELECT *
2  FROM R1 INNER JOIN R2
3  ON <condition>
```

Et pour plusieurs relations :

```
1  SELECT *
2  FROM (R1 INNER JOIN R2 ON <condition>) INNER JOIN Ri ON <condition>
3
```

Exemple

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	Delta	

R2

#X	Y
10	Echo
20	Fox
30	Golf

R

A	B	C	X	Y
1	Alpha	10	10	Echo
2	Bravo	10	10	Echo
3	Charlie	20	20	Fox

```
Jointure(R1,R2,R1.C=R2.X)

SELECT *
FROM R1 INNER JOIN R2
ON R1.C=R2.X
```

Tableau 36 Exemple de jointure (SQL et Algèbre)

Exemple : Une jointure naturelle

```
1  SELECT *
2  FROM R1, R2
3  WHERE R2.NUM = R1.NUM
```

Remarque : Auto-jointure

Pour réaliser une auto-jointure, c'est à dire la jointure d'une relation avec elle-même, on doit utiliser le renommage des relations. Pour renommer une relation, on note dans la clause FROM le nom de renommage après le nom de la relation :

"FROM NOM_ORIGINAL NOUVEAU_NOM".

Exemple : Auto-jointure

```
1  SELECT E1.Nom
2  FROM Employe E1, Employe E2
3  WHERE E1.Nom= E2.Nom
```

e) Exercice : Tableau final

Que renvoie la dernière instruction SQL de la séquence ci-dessous ?

```
1  CREATE TABLE R (a INTEGER, b INTEGER);
2  INSERT INTO R VALUES (1,1);
3  INSERT INTO R VALUES (1,2);
4  INSERT INTO R VALUES (3,3);
5  SELECT * FROM R R1, R R2 WHERE R1.a=R2.a;
```

1		1
1		2
3		3

1		1
1		1
1		2
1		2
3		3
3		3

1	1	1	1
1	1	1	2
1	1	3	3
1	2	1	1
1	2	1	2
1	2	3	3
3	3	1	1
3	3	1	2
3	3	3	3

1	1	1	1
1	1	1	2
1	2	1	1
1	2	1	2
3	3	3	3

1	1	1	2
1	1	3	3
1	2	1	1
1	2	3	3
3	3	1	1
3	3	1	2

f) Expression d'une jointure externe

Syntaxe : Jointure externe, gauche ou droite

Pour exprimer une jointure externe on se base sur la syntaxe INNER JOIN en utilisant à la place OUTER JOIN, LEFT OUTER JOIN ou RIGHT OUTER JOIN.

Exemple : Jointure externe gauche

```
1  SELECT Num
2  FROM Avion LEFT OUTER JOIN Vol
3  ON Avion.Num=Vol.Num
```

Cette requête permet de sélectionner tous les avions, y compris ceux non affectés à un vol.

Remarque

Remarquons que "Avion LEFT OUTER JOIN Vol" est équivalent à "Vol RIGHT OUTER JOIN Avion" en terme de résultat.

Intuitivement, on préfère utiliser la jointure gauche pour sélectionner tous les tuple du côté N d'une relation 1:N, même si il ne sont pas référencés ; et la jointure droite pour pour sélectionner tous les tuples d'une relation 0:N, y compris ceux qui ne font pas de référence. Cette approche revient à toujours garder à gauche de l'expression "JOIN" la relation "principale", i.e. celle dont on veut tous les tuples, même s'ils ne référencent pas (ou ne sont pas référencés par) la relation "secondaire".

Exemple

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	Delta	

R2

#X	Y
10	Echo
20	Fox
30	Golf

R

A	B	C	X	Y
1	Alpha	10	10	Echo
2	Bravo	10	10	Echo
3	Charlie	20	20	Fox

```
Jointure(R1,R2,R1.C=R2.X)

SELECT *
FROM R1 INNER JOIN R2
ON R1.C=R2.X
```

Tableau 37 Exemple de jointure (SQL et Algèbre)

Exemple

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	**Delta**	

R2

#X	Y
10	Echo
20	Fox
30	Golf

R

A	B	C	X	Y
1	Alpha	10	10	Echo
2	Bravo	10	10	Echo
3	Charlie	20	20	Fox
4	**Delta**			

```
JointureExterneGauche(R1,R2,R1.C=R2.X)

SELECT *
FROM R1 LEFT OUTER JOIN R2
ON R1.C=R2.X
```

Tableau 38 Exemple de jointure externe gauche (SQL et Algèbre)

Exemple

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	Delta	

R2

#X	Y
10	Echo
20	Fox
30	**Golf**

R

A	B	C	X	Y
1	Alpha	10	10	Echo
2	Bravo	10	10	Echo
3	Charlie	20	20	Fox
			30	**Golf**

```
JointureExterneDroite(R1,R2,R1.C=R2.X)

SELECT *
FROM R1 RIGHT OUTER JOIN R2
ON R1.C=R2.X
```

Tableau 39 Exemple de jointure externe droite (SQL et Algèbre)

Méthode : Trouver les enregistrements non joints

La jointure externe sert en particulier pour trouver les enregistrements d'une table qui ne sont pas référencés par une clé étrangère. Il suffit de sélectionner, après la jointure externe, tous les enregistrements pour lesquels la clé de la relation référençante est nulle, on obtient alors ceux de la relation référencée qui ne sont pas référencés.

R1

#A	B	C=>R2
1	Alpha	10
2	Bravo	10
3	Charlie	20
4	Delta	

R2

#X	Y
10	Echo
20	Fox
30	**Golf**

R

A	B	C	X	Y
			30	**Golf**

```
Restriction(
        JointureExterneDroite(R1,R2,R1.C=R2.X),
        R1.A IS NULL)

SELECT *
FROM R1 RIGHT OUTER JOIN R2
ON R1.C=R2.X
WHERE R1.A IS NULL
```

Tableau 40 Exemple de sélection d'enregistrements non référencés (SQL et Algèbre)

g) Exercice : Photos à gauche

Soit les trois relations instanciées suivantes :

Numero	Nom	Prenom
1	Jacques	Brel
2	Georges	Brassens
3	Léo	Ferré

Tableau 41 Relation Personne

Personne	Photo
1	1
2	1
3	1

Tableau 42 Relation PersonnesPhotos

Numero	Date	Lieu
1	10/12/1965	Paris
2	18/03/2002	Lille

Tableau 43 Relation Photo

Combien de tuples renvoie l'instruction SQL suivante :

```
1 SELECT *
2 FROM Photo ph
3 LEFT JOIN PersonnesPhotos pp ON ph.Numero=pp.Photo
4 LEFT JOIN Personne pe ON pp.Personne=pe.Numero;
```

h) Opérateurs ensemblistes

Introduction

Les opérateurs ensemblistes ne peuvent être exprimés à l'aide de l'instruction de sélection seule.

Syntaxe : Union

```
1 SELECT * FROM R1
2 UNION
3 SELECT * FROM R2
```

Syntaxe : Intersection

```
1 SELECT * FROM R1
2 INTERSECT
3 SELECT * FROM R2
```

Syntaxe : Différence

```
SELECT * FROM R1
EXCEPT
SELECT * FROM R2
```

Remarque

Les opérations INTERSECT et EXCEPT n'existe que dans la norme SQL2, et non dans la norme SQL1. Certains SGBD sont susceptibles de ne pas les implémenter.

B. Exercices

1. Location d'appartements

[30 min]

Soit le schéma relationnel suivant gérant le fonctionnement d'une agence de location d'appartements.

```
APPARTEMENT(#code_appt:String, adresse:String, type:
 {studio,F1,F2,F3,F4,F5+}, prix_loyer:Real)
LOCATAIRE(#code_loc:String, nom:String, prenom:String)
LOCATION(#code_loc=>Locataire, #code_appt=>Appartement)
PAIEMENT_LOYER(#code_loc=>Locataire, #code_appt=>Appartement,
 #date_payement:Date, prix_paye:Real)
```

Question 1

En algèbre relationnelle et en SQL afficher tous les paiements effectués par un locataire avec le code X.

Question 2

En algèbre relationnelle et en SQL afficher l'adresse de l'appartement loué par un locataire avec le code X.

Question 3

En algèbre relationnelle et en SQL proposer une requête qui affiche tous les appartements libres.

2. Employés et salaires

[20 minutes]

Soit le schéma relationnel :

```
Employe (#Num, Nom, Prenom, Age, Salaire, Fonction=>Fonction,
 Societe=>Societe)
Fonction (#Intitule, SalaireMin, SalaireMax, NbHeures)
Societe (#Nom, Pays, Activite)
```

Question 1

Écrivez une requête SQL permettant de sélectionner les noms de tous les directeurs de France.

Question 2

Écrivez une requête SQL permettant d'afficher le salaire de tous les employés en francs (sachant que le salaire dans la table est en euros et que un euro vaut 6.55957 franc).

Question 3

Écrivez une requête SQL permettant de vérifier que les salaires de chaque employé correspondent bien à ce qui est autorisé par leur fonction.

Question 4

Écrivez une requête SQL permettant le passage aux 35 heures :

- en modifiant le nombre d'heures travaillées pour chaque fonction (ramené à 35 s'il est supérieur à 35),
- et en réajustant les échelles de salaire au pro-rata du nombre d'heures travaillées avant le passage aux 35 heures.

III

Expression de contraintes en UML et en relationnel

A. Cours

1. Expression de contraintes sur le diagramme de classes

Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

a) Exercice : Modéliser un site Web

En analysant le schéma UML ci-après, sélectionner toutes les assertions vraies.

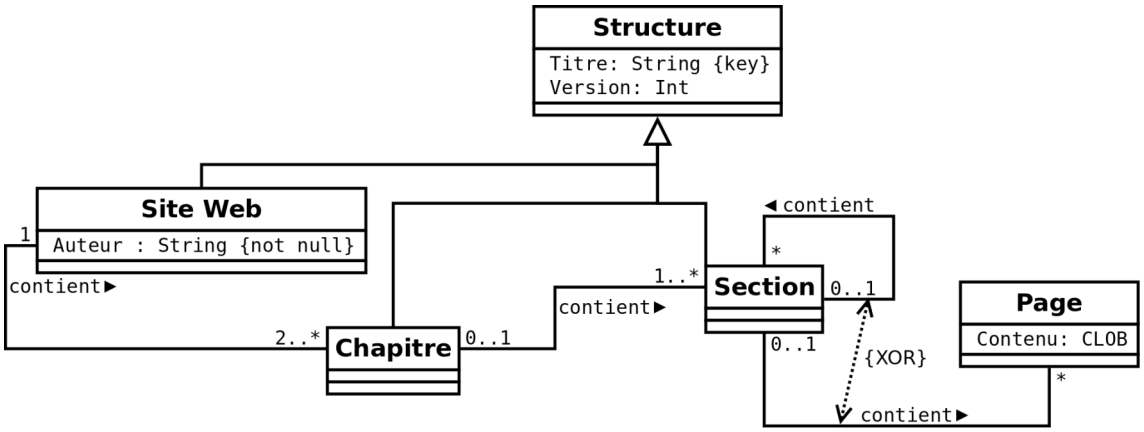


Image 1 MCD UML

- ☐ Tous les chapitres ont un titre.
- ☐ Il est possible d'avoir un auteur différent pour chaque chapitre.
- ☐ Toutes les sections contiennent au moins une section.
- ☐ Toutes les sections contiennent au moins une page.
- ☐ Toutes les sites Web contiennent au moins une page.

b) Contraintes

Ajout de contraintes dynamiques sur le diagramme de classe

Il est possible en UML d'exprimer des contraintes dynamiques sur le diagramme de classe, par annotation de ce dernier.

Syntaxe : Notation de contraintes

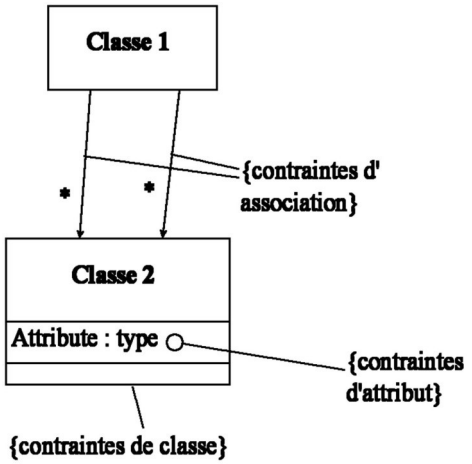


Image 2 Notation contraintes en UML

Exemple : Exemple de contraintes

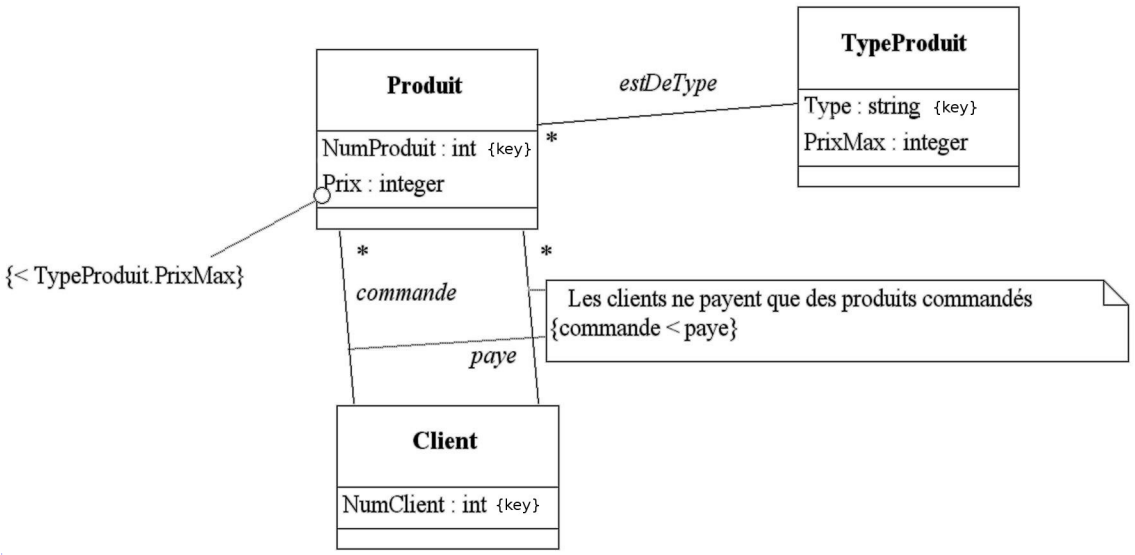


Image 3 Commandes

Méthode : Expressions formelles ou texte libre ?

Il est possible d'exprimer les contraintes en texte libre ou bien en utilisant des expressions formelles.

On privilégiera la solution qui offre le meilleur compromis entre facilité d'interprétation et non ambiguïté. La combinaison des deux est également possible si nécessaire.

Méthode : Quelles contraintes exprimer ?

En pratique il existe souvent de très nombreuses contraintes, dont certaines sont évidentes, ou encore secondaires. Or l'expression de toutes ces contraintes sur un diagramme UML conduirait à diminuer considérablement la lisibilité du schéma sans apporter d'information nécessaire à la compréhension. En conséquence on ne représentera sur le diagramme que les contraintes les plus essentielles.

Notons que lors de l'implémentation toutes les contraintes devront bien entendu être exprimées. Si l'on souhaite préparer ce travail, il est conseillé, lors de la modélisation logique, de recenser de façon exhaustive dans un tableau toutes les contraintes à implémenter.

c) Expression de l'optionalité

Syntaxe : Optionalité

L'attribut a **toujours** une valeur au cours de son cycle de vie, donc dès la création de l'objet auquel il appartient.

Syntaxe : Optionalité : [0..1]

On considère par défaut en UML que tous les attributs sont non nuls et on indique les attributs optionnels avec la syntaxe `attribut [0..1]`.

Syntaxe : Non nullité {not null}

Dans un contexte où tous les attributs seraient optionnels sauf exception, on peut

adopter l'expression de la non nullité {not null} à la place de l'expression de l'optionalité. On n'utilisera pas les deux syntaxes sur un même schéma UML (sous peine de ne pas savoir comment interpréter les attributs sans [0..1] ni {not null}.

d) Expression de l'unicité

Définition : Unicité

La valeur de l'attribut est unique pour la classe.

Syntaxe : {unique}

On utilise {unique} pour exprimer l'unicité.

Syntaxe : {unique} ou {key}

Bien que non standardisé en UML, en base de données, il est courant d'utiliser {key} à la place de {unique}.

- {key} ≡ {unique not null} (à condition que la minimalité soit respectée par ailleurs) ;
- et comme {not null} est une contrainte pas défaut : {unique} ≡ {key} (si l'optionalité n'est pas exprimée et si la contrainte de minimalité est vérifiée)

Exemple : Contrainte d'unicité

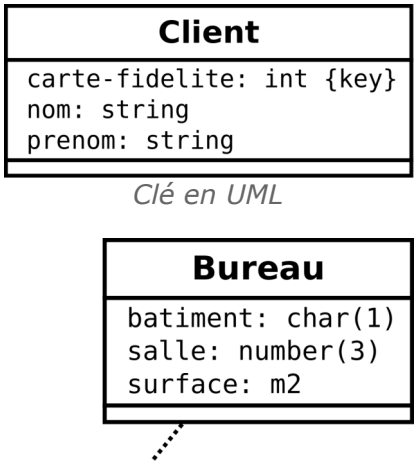


Image 4 Clé composée de deux attributs

e) Expression de l'immuabilité

Définition : Immuabilité

Une fois instancié l'attribut ne peut plus changer.

Définition : {frozen} ou {immuable}

On peut écrire {frozen} ou {immutable}.

Rappel : Clé primaire

Pour choisir une clé primaire au niveau relationnel, on privilégiera une clé candidate immuable, et donc une contrainte {unique frozen} (sur un groupe minimal

d'attributs non nuls).

f) Contraintes standardisées sur les associations

Exemple : Exemple de notation

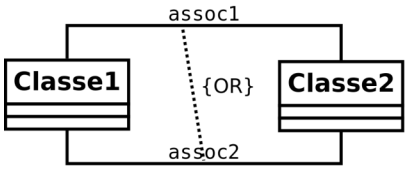


Image 5 Contrainte OR entre deux associations

Classe1	Classe2	IN(C1→C2)	IN(C2→C1)	AND ou S	X	OR ou T	XOR ou XT
0	0	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0

Définition : Inclusion {I}, également notée {Subset} ou {IN}

Si l'association inclue est instanciée, l'autre doit l'être aussi, la contrainte d'inclusion a un sens, représenté par une flèche.

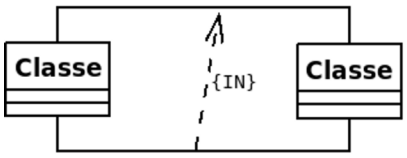


Image 6 Contrainte orientée IN

Définition : Simultanéité {S}, également notée {=} ou {AND}

Si une association est instanciée, l'autre doit l'être aussi.
La simultanéité est équivalente à une double inclusion.

Définition : Exclusion {X}

Les deux associations ne peuvent être instanciées en même temps.

Définition : Totalité {T}, également notée {OR}

Au moins une des deux associations doit être instanciée.

Définition : Partition {XT}, également notée {+} ou {XOR} ou {P}

Exactement une des deux associations doit être instanciée.

2. Passage UML-Relationnel : Expression de contraintes

Objectifs

Savoir exprimer les contraintes en modélisation relationnelle.

a) Liste des contraintes

Fondamental : KEY, UNIQUE, NOT NULL

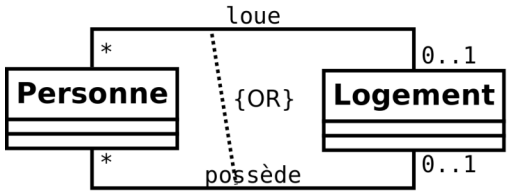
On pensera à exprimer les clés candidates KEY, les attributs UNIQUE et NOT NULL.

```
1  R1 (#pk, k, u, a) avec k KEY, u UNIQUE, a NOT NULL
```

Méthode : Contraintes exprimées sur le MCD

Les contraintes exprimées au niveau conceptuel doivent être traduites au niveau relationnel.

Si les contraintes ne sont pas trop nombreuses, on peut commenter chaque relation directement lors de la formalisation du MLD.



Exemple de contrainte OR entre deux associations

```
1  Personne (... loue=>Logement, possède=>Logement) avec (loue OR
   possède)
2  Logement (...)
```

Méthode

Si l'expression des contraintes nuit à la lisibilité, on les reportera une liste à la fin du MLD (voire dans un document annexe qui accompagnera le modèle logique).

Attention : Extension des contraintes exprimées

On s'attachera lors de la modélisation logique à exprimer l'ensemble des contraintes pesant sur le modèle, même celles qui ont été considérées comme secondaires ou évidentes lors de la modélisation conceptuelle.

Méthode : Expression des contraintes

Pour exprimer les contraintes on favorisera un langage formel dans la mesure du possible :

- algèbre relationnelle
- expression logique ou mathématique
- expression SQL ou procédurale
- ...

Remarque : Immutabilité

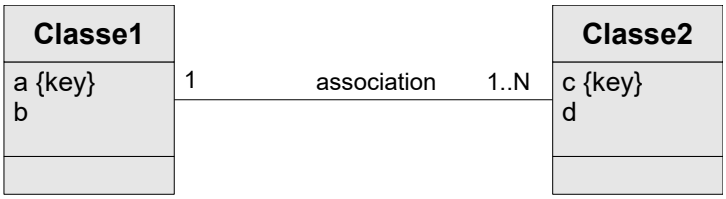
La contrainte {frozen} n'existe pas en base de données relationnelle standard. On ajoutera donc cette contrainte en complément dans le modèle relationnel (et on pourra ajouter un déclencheur au moment de l'implémentation pour empêcher la modification d'un attribut).

b) Contrainte de cardinalité minimale 1 dans les associations 1:N

Rappel

Transformation des associations 1:N

Méthode



Graphique 1 Association 1:N

- Si la cardinalité est exactement 1 (1..1) côté 1, alors on ajoutera une contrainte de non nullité sur la clé étrangère,
- si la cardinalité est au moins 1 (1..N) côté N, on ajoutera une contrainte d'existence de tuples référençant pour chaque tuple de la relation référencée.

`R1(#a,b)`

`R2(#c,d,a=>R1)`

Contraintes : `R2.a NOT NULL et PROJECTION(R1,a) = PROJECTION(R2,a)`

On veut que tous les éléments de R1 soient référencés au moins une fois dans R2, donc il n'existe pas de tuple de R1 qui ne soit pas référencé par R2.a, donc `PROJECTION(R1,a) = PROJECTION(R2,a)`.

#a	b
1	Lorem

#a	b
2	Ipsum

Tableau 44 R1

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2

Tableau 45 R2

Complément : Cas général : PROJECTION(R1,a) ⊆ PROJECTION(R2,a)

Si la cardinalité côté 1 est 0..1 alors il peut exister la valeur NULL dans R2.a et donc la contrainte devient : `PROJECTION(Classe1,a) ⊆ PROJECTION(Classe2,a)`.

#a	b
1	Lorem
2	Ipsum

Tableau 46 R1

#c	b	a=>R1
a	Sed	1
b	Ut	2
c	Perspiciatis	2
d	Unde	NULL

Tableau 47 R2

Rappel : La projection élimine les doublons

Projection

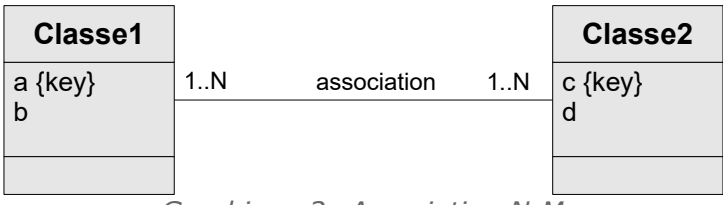
c) Contrainte de cardinalité minimale 1 dans les associations N:M

Rappel

Transformation des associations N:M

Méthode

Si la cardinalité est au moins 1 (1..N) d'un côté et/ou de l'autre, alors des contraintes d'existence simultanée de tuples devront être ajoutées.



Graphique 2 Association N:M

```
R1(#a,b)
R2(#c,d)
A(#a=>R1,#c=>R2)
```

Contraintes : `PROJECTION(R1,a) = PROJECTION(A,a) AND PROJECTION(R2,c) = PROJECTION(A,c)`

Remarque

Si la cardinalité est `0..N:1..N` seule une des deux contraintes est exprimée.

Rappel : La projection élimine les doublons

Projection

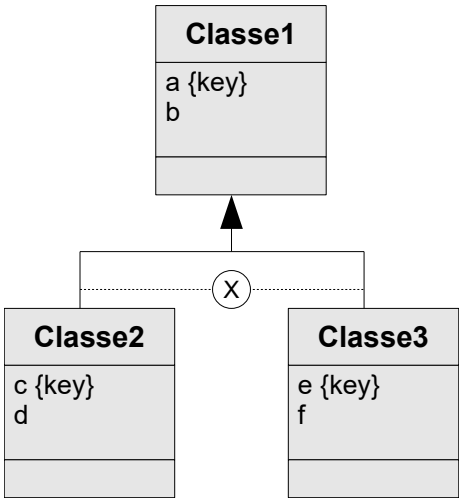
d) Contraintes de l'héritage par référence

Rappel

Transformation de la relation d'héritage par référence

Méthode : Héritage exclusif (cas général)

Dans le cas général d'un héritage exclusif, il faut ajouter une contrainte pour que les même tuples ne se trouvent pas à la fois dans R2 et R3



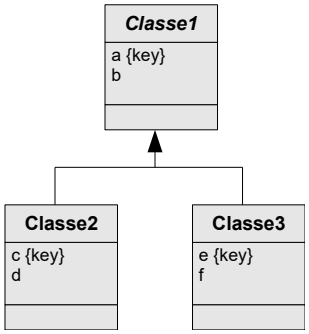
Graphique 3 Héritage exclusif (non complet)

```
R1(#a,b)
R2(#a=>R1,c,d) avec c KEY
R3(#a=>R1,e,f) avec e KEY
```

Contrainte : `INTERSECTION (PROJECTION(R2,a), PROJECTION(R3,a)) = {}`

Méthode : Héritage par référence avec classe mère abstraite

Si la classe mère est abstraite, il faut ajouter la contrainte que tous les tuples de R1 sont référencés par un tuple de R2 ou de R3.



Graphique 4 Héritage (classe mère abstraite)

```
R1(#a,b)
R2(#a=>R1,c,d) avec c KEY
R3(#a=>R1,e,f) avec e KEY
```

Contraintes : `PROJECTION(R1,a) = PROJECTION(R2,a) UNION PROJECTION(R3,a)`

Exemple

Soit la classe A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A, comprenant la clé K' et les attributs B1 et B2. Le modèle relationnel correspondant selon cette transformation est :

```
1  A (#K, A1, A2)
2  B (#K=>A, K', B1, B2)
```

Cette solution est particulièrement adaptée lorsque la classe mère n'est pas abstraite, car cela en autorise l'instanciation sans aucun bruit dans la relation lié aux classes filles. Par contre si la classe mère est abstraite il faut introduire une contrainte dynamique complexe pour imposer la présence d'un tuple référençant dans une des classes filles.

Ainsi dans l'exemple précédent, un A peut être instancié en toute indépendance par rapport à la relation B.

e) Contraintes de l'héritage par les classes filles

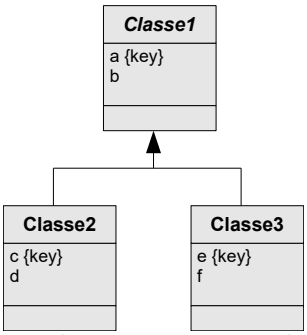
Rappel

Transformation de la relation d'héritage par les classes filles

Méthode : Héritage par les classes filles avec clé au niveau de la classe classe mère

Dès lors que la classe mère possède un attribut unique :

- on ajoute une contrainte pour vérifier cette unicité au niveau des classes filles.



Graphique 5 Héritage (classe mère abstraite)

R2(#a,b,c,d) avec c KEY

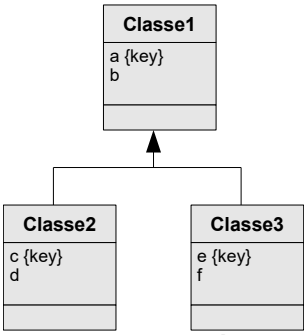
R3(#a,b,e,f) avec e KEY

Contraintes : INTERSECTION (PROJECTION(R2,a), PROJECTION(R3,a)) = {}

Méthode : Héritage par les classes filles avec classe mère non abstraite

Si la classe mère n'est pas abstraite :

- On créé une relation supplémentaire pour gérer les objets de la classe mère
- On ajoute une contrainte qui exprime que les tuples de la classe mère ne peuvent pas exister dans les classes filles



Graphique 6 Héritage

R1(#a,b)

R2(#a,b,c,d) avec c KEY

R3(#a,b,e,f) avec e KEY

Contraintes :

- INTERSECTION (PROJECTION(R2,a), PROJECTION(R3,a)) = {}
- INTERSECTION (PROJECTION(R1,a), (PROJECTION(R2,a) UNION PROJECTION(R3,a)) = {}

Exemple : Héritage absorbé par les classes filles

Soit la classe abstraite A avec la clé K et les attributs A1 et A2. Soit la classe B, classe fille de A avec les attributs B1 et B2. Soit la classe C, classe fille de A avec les attributs C1 et C2.

Le modèle relationnel correspondant selon cette transformation est :

```
1  B (#K, A1, A2, B1, B2)
2  C (#K, A1, A2, C1, C2)
3  Contrainte : INTERSECTION (PROJECTION(B,K), PROJECTION(C,K)) = {}
```

Si A n'avait pas été abstraite elle aurait donné naissance à une relation A possédant

les attributs A1 et A2 et qui n'aurait alors contenu que les tuples qui ne sont ni des B ni des C.

f) Contraintes de l'héritage par la classe mère

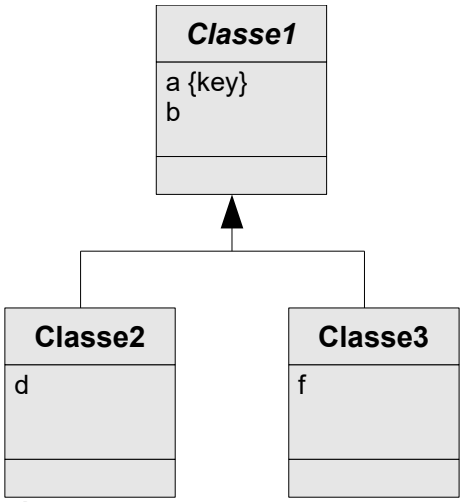
Rappel

Transformation de la relation d'héritage par la classe mère

Méthode : Héritage non complet

Si l'héritage n'est pas complet :

- il faut vérifier la cohérence du type avec les attributs valués
- il faudra gérer la non nullité « à la main »



Graphique 7 Héritage presque complet (classe mère abstraite)

`R1(#a,b,c,d,e,f,t:{2,3})`

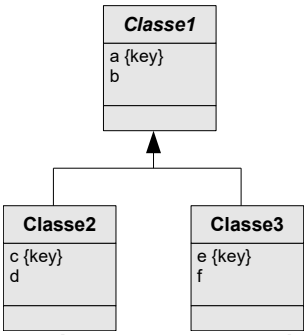
Contraintes :

- `t NOT NULL`
- `NOT (t=2 AND f)`
- `NOT (t=3 AND d)`
- `t=2 AND d` (si d est non nul)
- `t=3 AND f` (si f est non nul)

Méthode : Héritage par la classe mère avec classe mère abstraite

Si la classe mère est abstraite :

1. sa valeur est ôtée de l'attribut de discrimination ;
2. une contrainte supplémentaire doit vérifier que soit c soit e est obligatoirement valué.



Graphique 8 Héritage (classe mère abstraite)

```
R1(#a,b,c,d,e,f,t:{2,3})
```

Contraintes :

- `t NOT NULL`
- `c UNIQUE`
- `e UNIQUE`
- `t=2 AND c`
- `t=3 AND e`
- `NOT (t=2 AND (e OR f)`
- `NOT (t=3 AND (c OR d)`

g) Contraintes simples et contraintes compliquées

Les contraintes exprimées dans le cadre de la conception de bases de données peuvent être catégorisées en trois types, du plus simple au plus complexe.

Fondamental

Il faut préférer les contraintes plus simples à celles plus compliquées dans la mesure où elles permettent d'obtenir le modèle recherché (principe du rasoir d'Ockham).

Cela minimisera le travail d'implémentation et de maintenance, et augmentera la fiabilité.

Les contraintes exprimables directement en relationnel

Ce sont les plus simples :

- UNIQUE
- NOT NULL

Les contraintes facilement réalisables en SQL (qui ne concernent qu'une seule relation)

Elles peuvent être exprimées directement en SQL au sein du CREATE TABLE via un CHECK.

Les contraintes compliquées qui nécessitent une couche de programmation en plus du SQL

- On favorisera l'expression de vues (permettant de détecter les anomalies typiquement).
- On utilisera les déclencheurs ou la couche applicative.

B. Exercices

1. Médiathèque

[60 minutes]

Une médiathèque veut s'informatiser pour gérer plus efficacement son catalogue, contenant des livres, des films, et des albums de musique.

Les informations à enregistrer pour les livres sont : les titres, le (ou les) auteur(s), la date de parution, l'éditeur, la date d'impression et le nombre de pages. Les informations à enregistrer pour les films sont : le nom, le (ou les) réalisateur(s), au maximum six acteurs ayant joué dans le film avec leur rôle (il peut n'y avoir aucun acteur), le studio de production et la date de sortie. Les informations à gérer pour les albums sont : le nom de l'album, l'auteur, la maison de production, l'année de sortie, et les supports disponibles à la médiathèque (CD et/ou vinyle).

Pour chaque auteur, réalisateur, acteur ou musicien d'une œuvre, on veut gérer son nom, son prénom, son ou ses noms d'artiste (s'il en a), et sa date de naissance. Un album de musique peut avoir pour auteur :

- un artiste (ex : Jimmy Hendrix, Mozart),
- ou un groupe (ex : les Rolling Stones ou Franz Ferdinand) dont on gérera le nom, l'année de formation, et les informations sur les différents membres.

Livres, films et albums sont identifiés de manière unique (un film ne peut pas avoir le même identifiant qu'un album) par un code interne à la médiathèque. On veut aussi pouvoir gérer les adaptations cinématographiques (quels films sont les adaptations de quels livres). On veut enfin pouvoir retrouver facilement la BO correspondant à un film donné (la bande originale est l'album de musique du film).

Question 1

Réalisez un *package* permettant de modéliser en UML le catalogue de la médiathèque.

La médiathèque s'étend sur trois étages, chacun comprenant plusieurs rayons à thème (Romans Contemporains, Musique Classique, etc.). Certains de ces rayons peuvent comprendre à la fois des albums, des livres et des films (comme le rayon Science Fiction, ou le rayon Thriller). Dans un rayon, les œuvres centrales ou récentes sont disposées sur des présentoirs (chacun pouvant recueillir un certain nombre d'œuvres : films, livres et/ou albums), tandis que les autres sont rangées sur des étagères identifiées par un code particulier. Chaque étagère peut comprendre un certain nombre d'œuvres, mais d'une seule catégorie (on a des étagères de livres et des étagères d'albums, mais pas d'étagères comprenant à la fois des livres et des albums).

Question 2

Réalisez un second *package* permettant de modéliser l'organisation de la bibliothèque et de ses moyens de rangement.

Modélisation avancée en UML et en relationnel

IV

A. Cours

1. Modélisation avancée des associations en UML

a) Agrégation

Définition : Association d'agrégation

L'agrégation est une association particulière utilisée pour préciser une relation tout/partie (ou ensemble/élément), on parle d'association **méréologique**.

Syntaxe

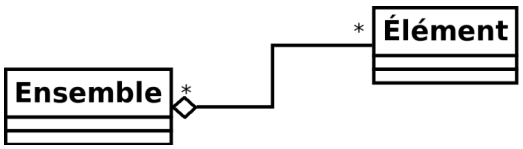


Image 7 Notation de l'agrégation en UML

La cardinalité, peut être exprimée librement, en particulier les instances de la classe Élément peuvent être associées à plusieurs instances de la classe Ensemble, et même de plusieurs classes.

Attention

L'agrégation garde toutes les propriétés d'une association classique (cardinalité, cycle de vie, etc.), elle ajoute simplement une terminologie un plus précise via la notion de tout/partie.

Rappel : Transformation des agrégations en relationnel

Les associations de type agrégation se traitent de la même façon que les associations classiques.

Graphique 9 Agrégation 1:N

```
Classe1(#a,b)
Classe2(#c,d,a=>Classe1)
```

Graphique 10 Agrégation N:M

```
Classe1(#a,b)
Classe2(#c,d)
Assoc(#a=>Classe1,#c=>Classe2)
```

b) Explicitation des associations (sens de lecture et rôle)

Syntaxe : Sens de lecture

Il est possible d'ajouter le sens de lecture du verbe caractérisant l'association sur un diagramme de classe UML, afin d'en faciliter la lecture. On ajoute pour cela un signe < ou > (ou un triangle noir) à côté du nom de l'association

Syntaxe : Rôle

Il est possible de préciser le rôle joué par une ou plusieurs des classes composant une association afin d'en faciliter la compréhension. On ajoute pour cela ce rôle à côté de la classe concernée (parfois dans un petit encadré collé au trait de l'association.

Exemple

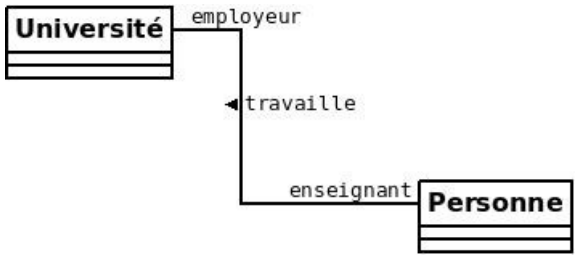


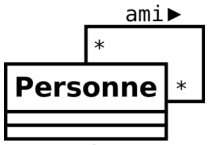
Image 8 Rôle et sens de lecture sur une association

c) Associations réflexives

Définition : Association réflexive

Une association réflexive est une association qui associe une classe avec elle-même.

Exemple



Association réflexive « ami »

Méthode

L'explicitation des associations est souvent utile dans le cas des associations réflexives non symétrique (ou chaque objet ne joue pas le même rôle).

Attention : Auto-association dans les associations réflexives

Une instance peut être associée avec elle-même dans le cas de d'une association réflexive.

Si l'on souhaite exprimer le contraire (une instance peut être associée avec d'autres instances de la même classe, mais pas avec elle-même) :

- on ajoute une contrainte en UML (par exemple {les personnes ne se marient pas avec elles-mêmes}) ;
- que l'on traduira en relationnel par une contrainte du type `AVEC pk ≠ fk` ;
- que l'on traduira en SQL par une clause du type `CHECK pk != fk`.

d) Classe d'association avec clé locale

Rappel

Classe d'association

Contrainte inhérente à la relation N:M

Dans l'exemple suivant, chaque personne peut avoir un emploi dans plusieurs sociétés, mais elle ne peut pas avoir plusieurs emplois dans une même société.

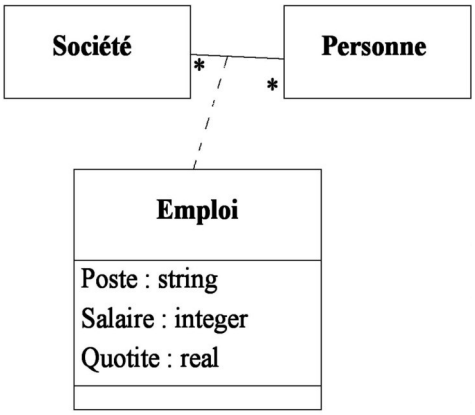


Image 9 Emplois

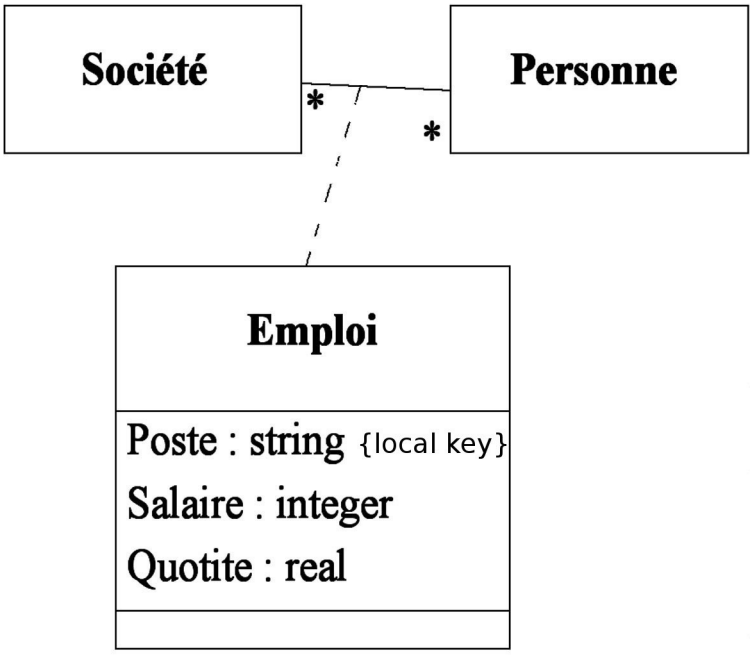
La transformation en relationnelle est cohérente avec cette contrainte.

```
1 Société(...)
2 Personne(...)
3 Emploi(#personne=>Personne, #societe=>Societe, poste:string,
  salaire:integer, quotite:numeric(1,2))
```

#personne	#societe	poste	salaire	quotite
~~Al~~	~~Canonical~~	Directeur	100000	0,5
~~Al~~	~~Canonical~~	Développeur	50000	0,5

Méthode : Intérêt de la clé locale

La spécification d'une clé locale dans emploi, par exemple ici le poste, permet de lever cette contrainte, lorsqu'on le souhaite, en permettant d'identifier chaque instance de l'association, ici l'emploi d'une personne par sa société.



La transformation en relationnelle permettra de maintenir la modélisation, en ajoutant la clé locale à la clé initiale

```
1 Société(...)
2 Personne(...)
3 Emploi(#personne=>Personne, #societe=>Societe, #poste:string,
  salaire:integer, quotite:numeric(7,2))
```

#personne	#societe	#poste	salaire	quotite
Al	Canonical	Directeur	100000	0,5
Al	Canonical	Développeur	50000	0,5

e) Exercice

Quel(s) schéma(s) relationnel(s) corresponde(nt) au modèle UML suivant (on ignore les domaines à des fins de simplification) ?

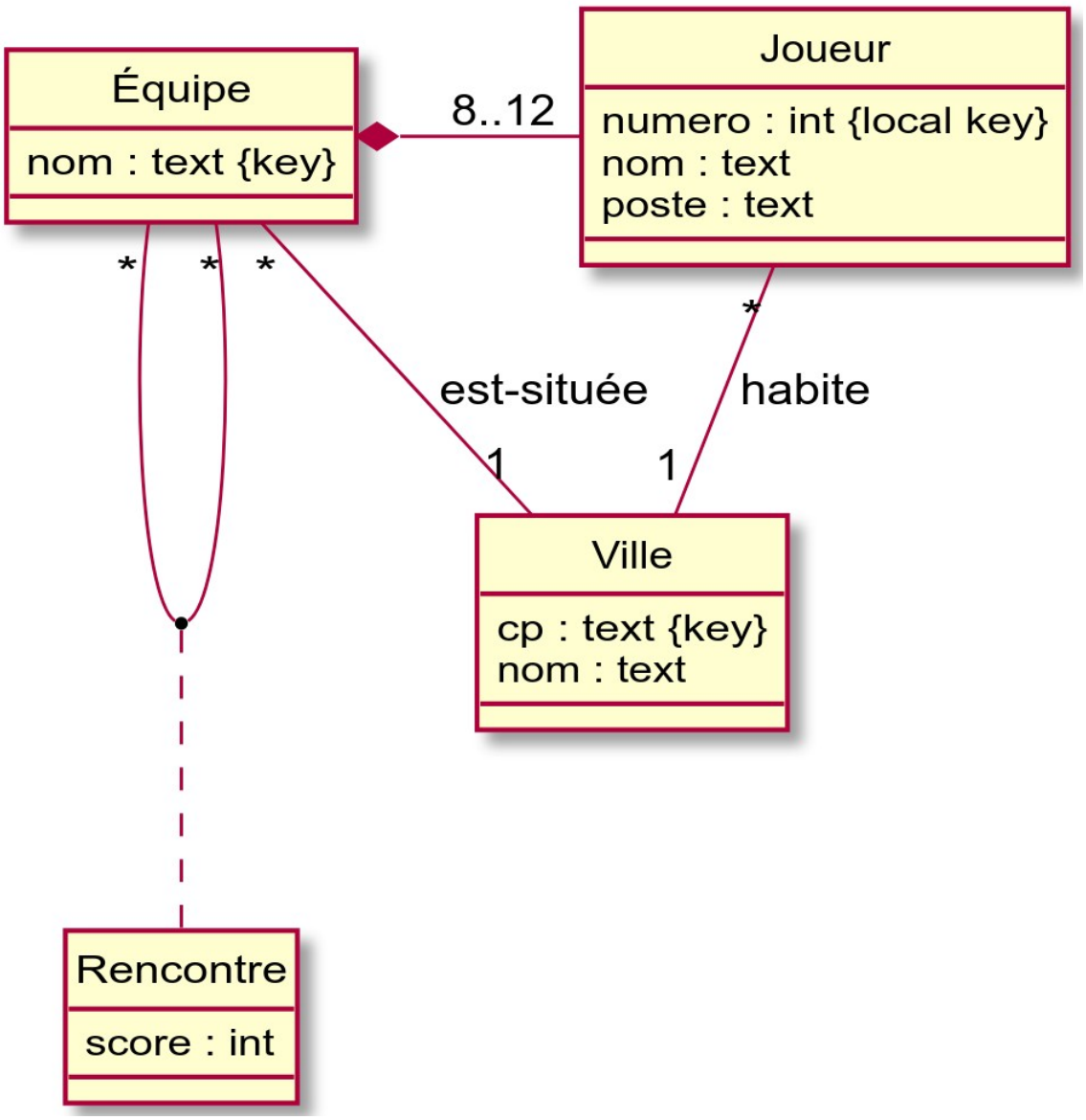


Image 10 Volley ball

☐	Joueur(#Numero, Nom, Ville=>Ville) Equipe(#Nom, Joueur=>Joueur, Poste, Rencontre=>Equipe, Score, Ville=>Ville) Vile(#CodePostal, Nom)

☐	Joueur(#Numero, #Equipe=>Equipe, Nom, Poste, Ville=>Ville)
	Equipe(#Nom, Ville=>Ville)
	Ville(#CodePostal, Nom)
	Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

☐	Joueur(#Numero, Nom, Ville=>Ville) Equipe(#Nom, Joueur=>Joueur, Poste, Ville=>Ville) Ville(#CodePostal, Nom) Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

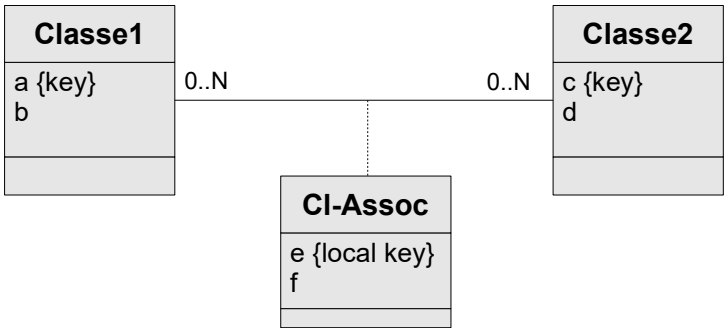
☐	Joueur(#Numero, Nom)
	Equipe(#Nom)
	Ville(#CodePostal, Nom)
	Habite(#Numero=>Joueur, #CodePostal=>Ville)
	Joue(#Numero=>Joueur, #Nom=>Equipe, Poste)
	EstSituee(#Nom=>Equipe, #CodePostal=>Ville)
	Rencontre(#NomE1=>Equipe, #NomE2=>Equipe, Score)

f) Transformation des classes d'association avec clé locale

Méthode : Classe d'association N:M

Les attributs de la classe d'association,

- sont ajoutés à la relation issue de l'association N:M ;
- la clé locale de la classe d'association est concaténée aux clés étrangères composant déjà la clé primaire de la relation d'association.



Graphique 11 Classe assocation (N:M)

```
Classe1(#a,b)
Classe2(#c,d)
Assoc(#a=>Classe1,#c=>Classe2,#e,f)
```

g) Associations ternaires

Syntaxe

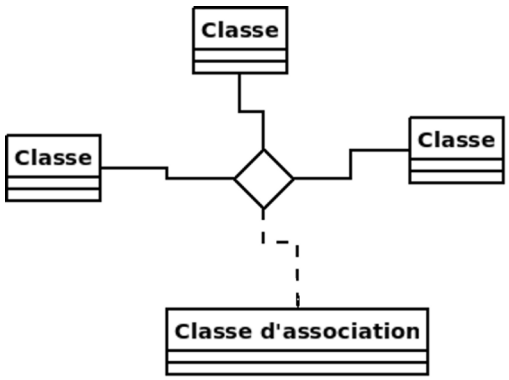


Image 11 Notation d'une association ternaire

Conseil : Ne pas abuser des associations ternaires

Il est toujours possible de réécrire une association ternaire avec trois associations binaires, en transformant l'association en classe.

Conseil : Pas de degré supérieur à 3

En pratique on n'utilise jamais en UML d'association de degré supérieur à 3.

2. Modélisation avancée des associations 1:1 en relationnel

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel.

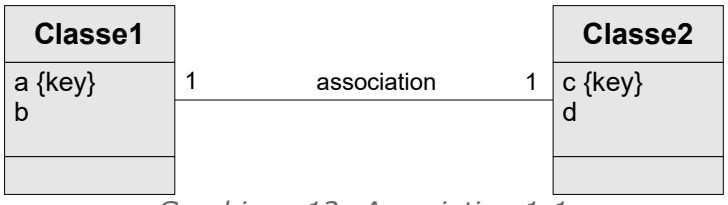
Connaître les choix possibles pour le cas de l'association 1:1 et savoir faire le meilleur choix.

Il existe des formulations conceptuelles en UML qui sont plus délicates à traduire au niveau logique en relationnel, comme l'association 1:1. Ces cas requièrent un choix éclairé de la part du concepteur.

a) Transformation des associations 1:1 (approche générale)

Il existe deux solutions pour transformer une association 1:1 :

- **Avec deux relations** : on traite l'association 1:1 comme une association 1:N, puis l'on ajoute une contrainte UNIQUE sur la clé étrangère pour limiter la cardinalité maximale à 1 ;
- **Avec une seule relation** : on fusionne les deux classes en une seule relation.



Graphique 12 Association 1:1

Méthode : Avec deux relations (clé étrangère)

- Une des deux relations est choisie pour porter la clé étrangère ;
- on ajoute les contraintes : `UNIQUE` ou `KEY` (clé candidate) sur la clé étrangère ; et si nécessaire une contrainte imposant l'instanciation simultanée des deux relations.

```
Classe1(#a,b,c=>Classe2) avec c UNIQUE ou KEY
Classe2(#c,d)
```

ou

```
Classe1(#a,b)
Classe2(#c,d,a=>Classe1) avec a UNIQUE ou KEY
```

Méthode : Avec une relation (fusion)

- On créé une seule relation contenant l'ensemble des attributs des deux classes ;
- on choisit une clé parmi les clés candidates.

```
Classe12(#a,b,c,d) avec c UNIQUE ou KEY
```

ou

```
Classe21(#c,d,a,b) avec a UNIQUE ou KEY
```

Remarque : Fusion des relations dans le cas de la traduction de l'association 1:1

Ce choix entre les deux méthodes sera conduit par une appréciation du rapport entre :

- La complexité introduite par le fait d'avoir deux relations là ou une suffit
- La pertinence de la séparation des deux relations d'un point de vue sémantique
- Les pertes de performance dues à l'éclatement des relations
- Les pertes de performance dues au fait d'avoir une grande relation
- Les questions de sécurité et de sûreté factorisées ou non au niveau des deux relations
- ...

Complément

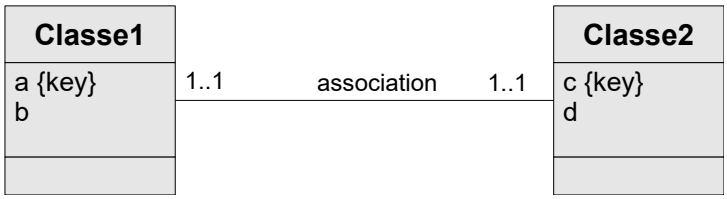
Dans le cas d'une association 1..1:1..1 il faudra ajouter une contrainte complémentaire.

Transformation des associations 1..1:1..1

b) Transformation des associations 1..1:1..1

Méthode : Association 1..1:1..1

- Le plus souvent c'est méthode par **fusion** des relations qui est la plus adaptée à ce cas.
- Lorsqu'elle ne l'est pas, et que l'on choisit deux relations il faut ajouter une contrainte dynamique qui contrôlera que les deux relations sont bien toujours instanciées ensembles. Notons que la clé étrangère peut être choisie comme clé primaire.



Graphique 13 Association 1:1

`Classe12(#a,b,c,d) avec c KEY`

ou

`Classe21(#c,d,a,b) avec a KEY`

ou

`Classe1(#a,b,c=>Classe2) avec c KEY`

`Classe2(#c,d)`

Contrainte : `PROJECTION(Classe1,c) = PROJECTION(Classe2,c)`

ou

`Classe1(#a,b)`

`Classe2(#c,d,a=>Classe1) avec a KEY`

Contrainte : `PROJECTION(Classe1,a) = PROJECTION(Classe2,a)`

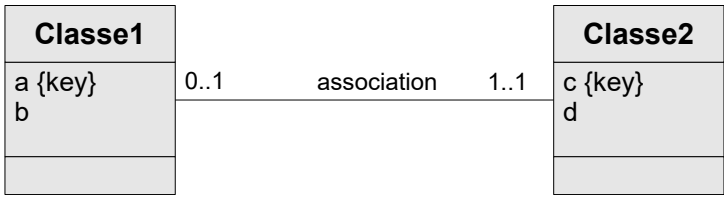
Complément : Contrainte dynamique

La contrainte dynamique exprimée ici suit le même principe que pour les association 1:N.*Contrainte de cardinalité minimale 1 dans les associations 1:N*

c) Transformation des associations 0..1:1..1

Méthode : Association 0..1:1..1

- Le plus souvent c'est la méthode par les deux relations qui est la plus adaptée, **on choisira toujours la relation côté 0..1 pour être référençante**.
- Il est possible d'utiliser la fusion, dans ce cas les clés côté 0..1 deviennent des attributs `UNIQUE`, il ne peuvent plus être clés, pouvant être `NULL`.



Graphique 14 Association 0..1:1

`Classe1(#a,b,c=>Classe2) avec c KEY`

`Classe2(#c,d)`

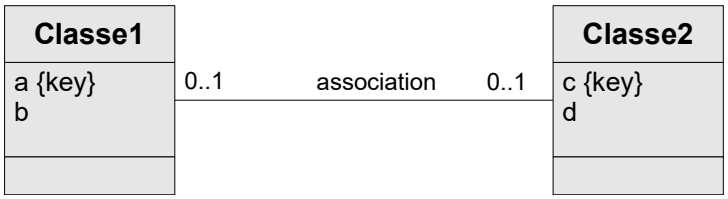
ou

```
Classe12(#c,d,a,b) avec a UNIQUE
```

d) Transformation des associations 0..1:0..1

Méthode : Association 0..1:0..1

- On choisit la solution avec deux relations, d'un côté ou de l'autre ; la clé étrangère est associé à la contrainte `UNIQUE`, ce n'est pas une clé car elle peut être nulle.
- Il n'est pas souhaitable de choisir la fusion, l'une des deux relations pouvant être nulle, on ne pourrait plus trouver de clé naturelle.



Graphique 15 Association 0..1:0..1

```
Classe1(#a,b,c=>Classe2) avec c UNIQUE
Classe2(#c,d)
```

ou

```
Classe1(#a,b)
Classe2(#c,d,a=>Classe1) avec a UNIQUE
```

e) Exemple de choix pour une relation 1:1

Exemple

- Soit deux entités, "homme" et "femme" et une association "mariage" de cardinalité 1..1:1..1 entre ces deux entités (hommes et femmes sont donc obligatoirement mariés).
- Les entités "homme" et "femme" sont identifiées par un attribut "nom" (dans ce modèle chaque personne a un nom unique, on pourrait remplacer le nom par un identifiant comme le numéro de sécurité social ou par une clé artificielle pour être plus proche de la réalité).

Bien que de type 1..1:1..1, le choix de la fusion n'est pas très opportun car il s'agit bien d'objets distincts que l'on veut modéliser.

- On choisira donc plutôt la représentation avec deux relations.
- La clé étrangère pourra être du côté "homme" ou "femme", même si la pratique dominante nous incite à la mettre du côté femme.
- On pourra également décider de prendre la clé étrangère comme clé primaire.

```
1  homme (#nom)
2  femme (#mariage=>homme, nom) avec nom KEY
3  Contrainte : PROJ(homme,nom) = PROJ(femme,mariage)
```

Exemple

Si l'association avait été de cardinalité 0..1:0..1 (certains hommes et femmes ne sont pas mariés), un choix similaire se serait imposé, avec l'impossibilité de choisir la clé étrangère comme clé primaire, celle-ci pouvant être nulle et n'étant donc plus candidate.

```
1  homme (#nom)
2  femme (#nom, mariage=>homme) avec mariage UNIQUE
```

3. Autres éléments utiles en UML : packages et stéréotypes

Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

a) Paquetages

Définition : Package

Les paquetages (plus communément appelés *package*) sont des éléments servant à organiser un modèle.

Ils sont particulièrement utiles dès que le modèle comporte de nombreuses classes et que celles-ci peuvent être triées selon plusieurs aspects structurants.

Syntaxe

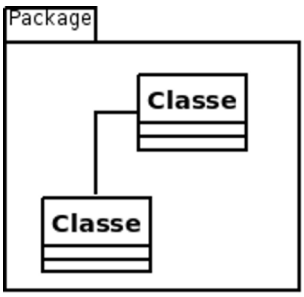


Image 12 Notation des paquetages en UML

Exemple

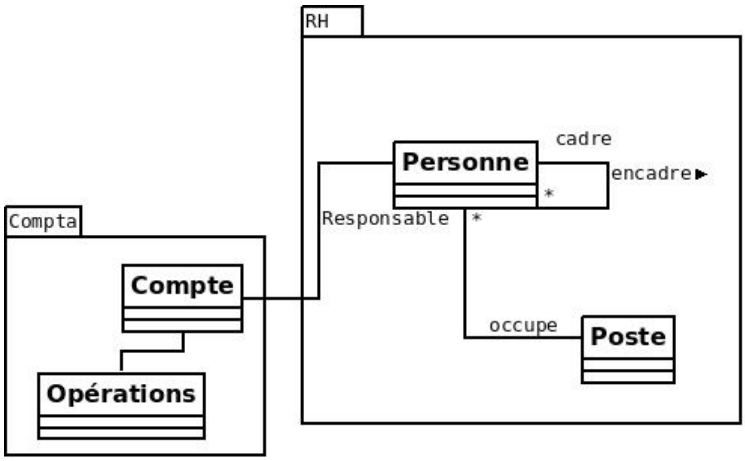


Image 13 Exemple d'utilisation des packages

Méthode

On représente chaque classe au sein d'un *package*. Il est alors possible de faire une présentation globale du modèle (tous les *packages*), partielle (une partie des *packages*) ou centrée sur un seul *package*.

Pour une représentation partielle ou centrée sur un *package*, on représente les *packages* concernés avec leurs classes propres, ainsi que toutes les classes liées des autres packages (et seulement celles-ci).

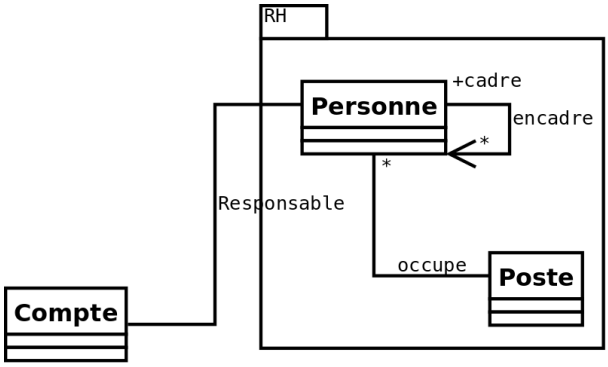


Image 14 Présentation partielle du modèle centrée sur un package

b) Stéréotype

Définition : Stéréotype UML

Un stéréotype UML est une syntaxe permettant d'ajouter de la sémantique à la modélisation des classes. Il permet de définir des **types de classe**, afin de regrouper conceptuellement un ensemble de classes (à l'instar d'une classe qui permet de regrouper conceptuellement un ensemble d'objets).

C'est une mécanique de méta-modélisation : elle permet d'étendre le méta-modèle UML, c'est à dire le modèle conceptuel du modèle conceptuel.

Définition : Méta-modèle

Un méta-modèle est le modèle d'un modèle. Par exemple le méta-modèle UML comprend les concepts de classe, attribut, association, cardinalité, composition, agrégation, contraintes, annotations, ... On mobilise ces concepts (on les instancie) pour exprimer un modèle particulier suivant le formalisme UML.

Les stéréotypes permettent donc d'ajouter au méta-modèle UML standard, celui que tout le monde utilise, des concepts locaux pour enrichir le langage de modélisation que l'on utilise pour réaliser des modèles.

Syntaxe

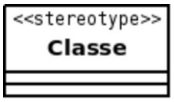


Image 15 Notation d'un stéréotype en UML

Conseil : Stéréotypes spécifiques et stéréotypes standard

Un stéréotype spécifique enrichit le méta-modèle UML, mais selon une sémantique qui est propre à celui qui l'a posé, non standard donc. La conséquence est que pour un tiers, l'interprétation du stéréotype n'est plus normalisée, et sera potentiellement plus facilement erronée. Il convient donc de ne pas abuser de cette

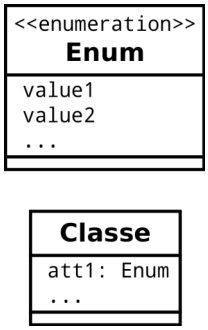
mécanique.

Deux ou trois stéréotypes spécifiques, correctement définis, sont faciles à transmettre, plusieurs dizaines représenteraient un nouveau langage complet à apprendre pour le lecteur du modèle.

Il existe des stéréotypes fournit en standard par UML, ou communément utilisés par les modélisateurs. L'avantage est qu'il seront compris plus largement, au même titre que le reste du méta-modèle (ils ont une valeur de standard).

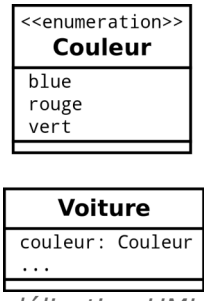
c) Énumération : stéréotype enumeration

Syntaxe



Stéréotype UML permettant d'exprimer une énumération

Exemple



Exemple de modélisation UML d'énumération

d) Quelques éléments de stylistique UML

Attention

- Toutes les associations doivent être nommées (sauf composition, héritage, agrégation).
- Ne pas utiliser de nom générique pour les Classes comme "Entité", "Classe", "Objet", "Truc"...
- Éviter les noms génériques pour les associations (comme "est associé à")
- Attention au sens des compositions et agrégation, le losange est côté ensemble, et n'oubliez pas les cardinalités côté parties.

Conseil

- N'utilisez pas le souligné ni les # en UML pour identifier les clés, préférez la contrainte {key}
- Préférez l'héritage aux booléens de typage en UML
- Les attributs dérivés sont réservés aux dérivations simples (des attributs de

la même classe), si c'est plus complexe, préférez des méthodes (et dans le doute, précisez les modes de calcul sur le schéma ou dans une note à part)

- Donnez des exemples de contenu lorsque ce n'est pas évident (lorsque le couple nom d'attribut et type ne permet pas de façon évidente de comprendre de quoi l'on parle)

e) Attention aux clés artificielles

Fondamental

- en UML : on ne pose jamais de clés artificielles
- en relationnel : on pose rarement des clé artificielles sauf dans le cas de clés étrangères vraiment trop compliquées
- en SQL : on peut poser des clés artificielles si on a une bonne raison (ce n'est donc pas systématique, et c'est à justifier)

Clés artificielles et niveau conceptuel

On n'ajoutera pas de clé artificielle en UML au moment de la modélisation conceptuelle des données.

1. **Formellement** en UML la notion de clé n'existe pas (contrairement à l'E-A), elle est ajoutée par les pratiquants des BD.
2. **Logiquement** on a pas besoin de cette notion en UML, les clés artificielles servent en relationnel et dans certains cas uniquement.
3. **Pratiquement** cela peut conduire à des situations absurdes (comme enlever au niveau logique des clés artificielles ajoutées au niveau conceptuel) :
   - si on fait du non-relationnel on ne doit pas ajouter de clés artificielles dans certains cas (l'imbrication typiquement) ;
   - cela arrive même en relationnel, avec la transformation de l'héritage (dans certains cas toujours).
4. **Méthodologiquement** il faut se concentrer à chaque phase sur ce qui est important, donc au moment du MCD on traduit les besoins, on repère les contraintes explicites (clé, unicité, non nullité...) sans se préoccuper de ce qui sera rendu nécessaire par la suite par la modélisation relationnelle (les clés étrangères par exemple) ou l'implémentation (l'optimisation par exemple). À chaque jour suffit sa peine !
5. **Pédagogiquement**, enfin, les "débutants" ont tendance (à cause des environnements de conception graphique, comme phpMyAdmin notamment) à systématiser les clés artificielles en SQL (on pourrait en discuter), mais également à ne pas faire le travail de recherche des clés naturelles (au niveau relationnel notamment, ce qui est une faute de modélisation). Donc au plus tard on fait intervenir les clés artificielles, au plus on a une chance de penser aux clés naturelles.

Clés artificielles et optimisation : est-il toujours plus performant d'utiliser des clés artificielles ?

Soit le modèle relationnel suivant :

```
1  Etu (#id, numEtu...) avec numEtu clé
2  UV (#id, codeUv...) avec codeUv clé
3  Inscriptions (#id, uv=>UV, etu=>Etu) avec (uv,etu) clé
```

La question qui permet d'afficher la liste des étudiants (numEtu) avec leurs UVs (codeUv) nécessite une jointure **à cause** des clés artificielles. Un modèle sans ces clés artificielles aurait été plus performant pour répondre à cette question, puisque toutes les informations se trouvent dans la relation `Inscriptions`.

```
1 Etu (#numEtu...)
2 UV (#codeUv...)
3 Inscriptions (#uv=>UV, #etu=>Etu)
```

Conseil

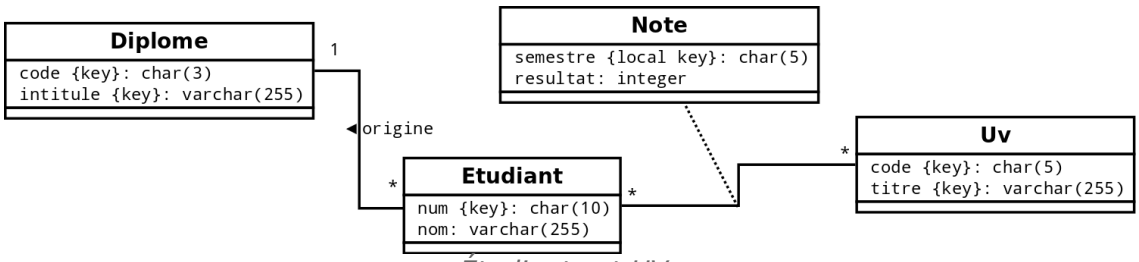
- Ne soyez pas systématique.
- Il est en effet fréquent dans un projet réel d'adopter des clés artificielles presque systématiquement, vous pourrez le faire en connaissance de cause quand vous aurez bien compris pourquoi c'est intéressant et quand ça ne l'est pas.
- Les clés artificielles sont intéressantes pour autre chose que les performances (l'évolutivité par exemple, choisissez-les quand vous savez pourquoi).
- Les clés artificielles ne dispensent pas de rechercher les clés naturelles !
- Les clés artificielles ne sont pas la seule façon d'optimiser une base de données (indexation, dénormalisation...)

B. Exercices

1. Étudiants et UVs (introduction)

[20 min]

On dispose du schéma UML ci-après qui décrit des étudiants, des UV, les notes obtenues par les étudiants à ces UV, et les diplômes d'origine de ces étudiants.



Étudiants et UVs

- {key} désigne des clés candidates, ici toutes les clés ne sont composées que d'un seul attribut
- {local key} désigne une clé locale
- un semestre est de la forme 'PYYYY' ou 'AYYYY' (où YYYY désigné une année sur 4 chiffre) ; exemple : 'A2013', 'P2014...

Rappel : *Notion de clé locale dans classes d'association*

Question

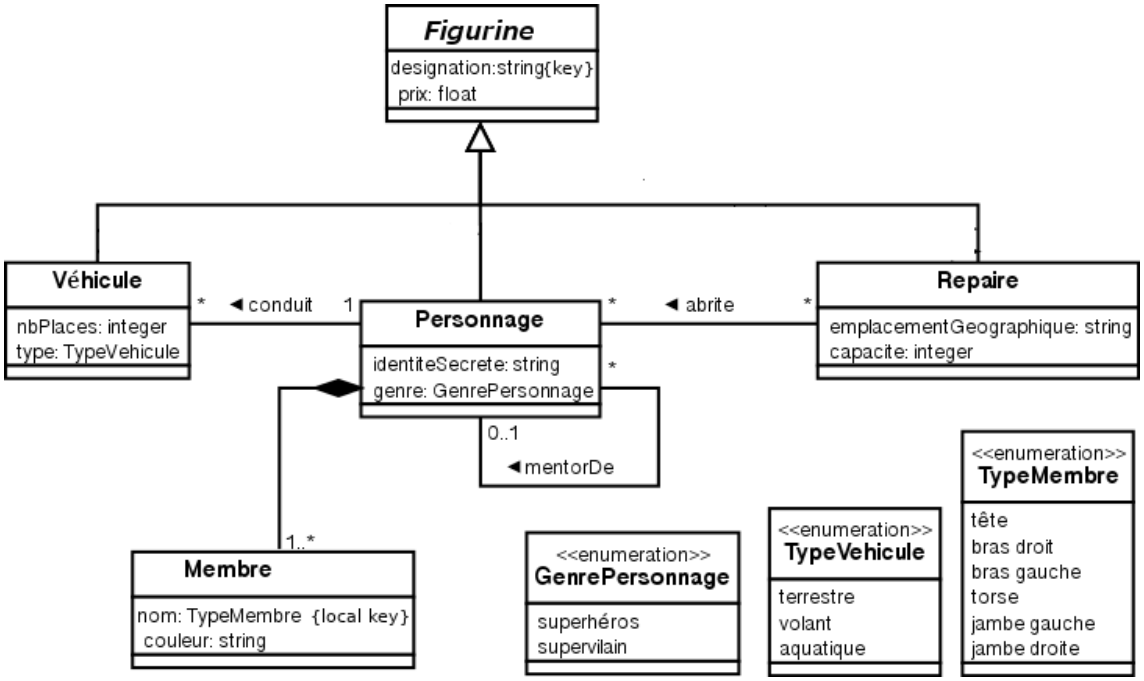
Traduire le schéma en modèle logique relationnel. (`MLD1`).

On choisira obligatoirement les clés primaires parmi celles nécessitant le plus petit nombre de bits possible pour leur codage.

2. Super-héros relationnels I

[20 min]

La gamme de super-héros GARVEL veut réaliser la base de données de leurs figurines articulées. La société a fait réaliser un modèle UML qui doit servir de point de départ à la mise en œuvre.



Modèle UML Figurines GARVEL

Question

Transformer le modèle UML en modèle relationnel (justifier les passages non triviaux, en particulier la relation d'héritage).

Imbrication en UML et en relationnel

V

A. Cours

1. Imbrication en UML : composition, attribut multivalué, datatype

Objectifs

Maîtriser le diagramme de classe UML dans le cas de la conception de BD.

a) Exercice : Entreprise

En analysant le schéma UML ci-après, sélectionner toutes les assertions vraies.

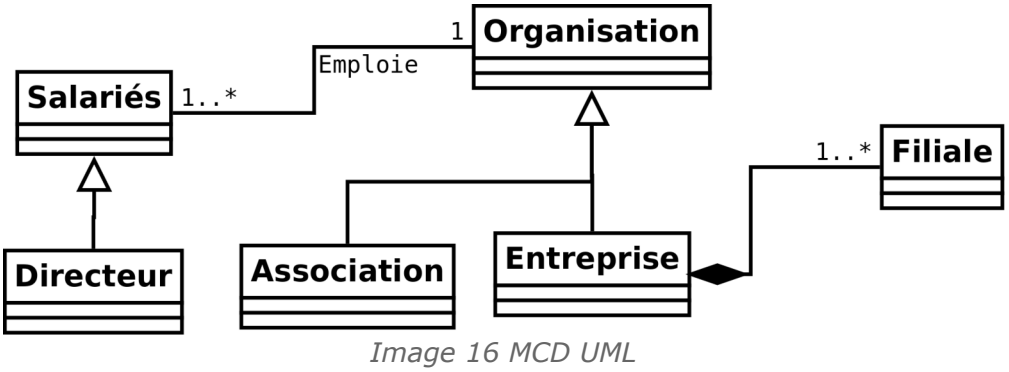


Image 16 MCD UML

☐	Une association peut employer un directeur.
☐	Une association peut employer plusieurs directeurs.
☐	Une association peut ne pas employer de directeur.
☐	Il existe des organisations qui ne sont ni des entreprises ni des associations.
☐	Une filiale peut appartenir à plusieurs entreprises.
☐	Une entreprise peut racheter la filiale d'une autre entreprise

b) Association de composition

Définition : Association de composition

On appelle composition une association particulière qui possède les propriétés suivantes :

- La composition associe une classe composite et des classes parties, tel que tout objet partie appartient à un et un seul objet composite. C'est donc une association 1:N (voire 1:1).
- La composition n'est pas partageable, donc un objet partie ne peut appartenir qu'à un seul objet composite à la fois.
- Le cycle de vie des objets parties est lié à celui de l'objet composite, donc un objet partie disparaît quand l'objet composite auquel il est associé disparaît.

Remarque

- La composition est une association particulière (binaire de cardinalité contrainte).
- La composition n'est pas symétrique, une classe joue le rôle de conteneur pour les classes liées, elle prend donc un rôle particulier a priori.
- La composition est une agrégation avec des contraintes supplémentaires (non partageabilité et cycle de vie lié).

Syntaxe : Notation d'une composition en UML

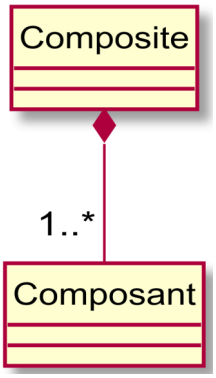


Image 17 Notation de la composition en UML

Exemple : Exemple de composition

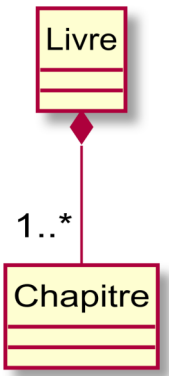


Image 18 Livre et chapitres

On voit bien ici qu'un chapitre n'a de sens que faisant partie d'un livre, qu'il ne peut exister dans deux livres différents et que si le livre n'existe plus, les chapitres le composant non plus.

Attention : Composition et cardinalité

La cardinalité côté composite est toujours de exactement 1.

Côté partie la cardinalité est libre, elle peut être 0..1, 1, * ou bien 1..*.

Attention : Composition et agrégation

L'agrégation (noté avec un losange blanc) est différent de la composition, puisqu'elle ne pose aucune dépendance au cycle de vie (c'est une association classique)

Complément : Composition et entités faibles

La composition permet d'exprimer une association analogue à celle qui relie une entité faible à une entité identifiante en modélisation E-A. L'entité de type faible correspond à un objet partie et l'entité identifiante à un objet composite.

Complément : Voir aussi

- *Agrégation*

c) Notion de clé locale

Le concept de clé locale appartient au niveau conceptuel, il est hérité de l'entité faible du modèle conceptuel Entité-Association (équivalent de la composition en UML). Dans une entité faible ou une composition, une clé de la classe composant est dite locale, car elle ne permet d'identifier l'objet que si l'on connaît la classe composite.

Définition

Dans certaines constructions en UML (association N:M et composition) la clé peut être locale, c'est à dire qu'au lieu d'identifier pleinement un objet (comme une clé classique), elle identifie un objet étant donné un contexte (les membres de l'association N:M ou l'objet composite).

Attention

Une clé locale n'est donc pas une clé au sens relationnel, elle ne permet pas d'identifier un enregistrement (mais elle deviendra une partie d'une clé lors du passage au relationnel).

d) Clé locale dans les compositions

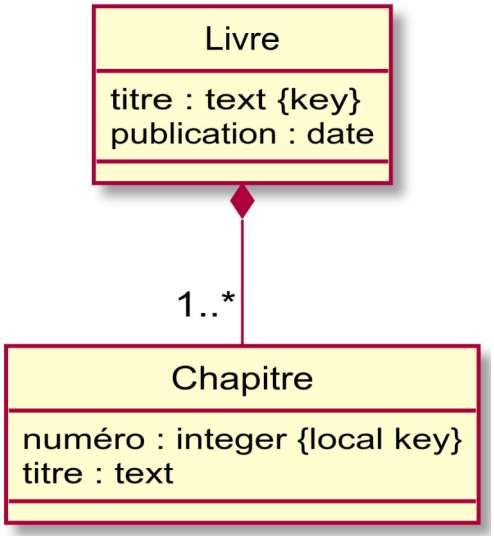
Rappel

Notion de clé locale

Attention

Dans une composition on a en général uniquement des clés locales : en effet si le composant est identifiable indépendamment de son composite, c'est en général qu'il a une vie propre et donc que l'on est pas en présence d'une composition.

Exemple



Livre et chapitres (avec clé locale)

Il existe deux plusieurs livres avec un chapitre numéro 1, mais il n'existe pas deux chapitre numéro 1 au sein d'un même livre.

e) Attribut multivalué

Définition

Un attribut multivalué est un attribut qui peut prendre plusieurs valeurs distinctes dans son domaine.

Syntaxe

```
1 attribut_mv[nbMinValeurs..nbMaxValeurs]:type
```

Exemple : La classe Personne

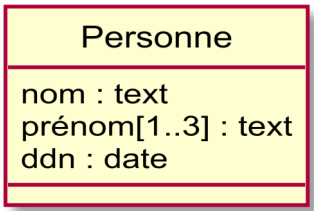


Image 19 Attribut multivalué

Dans cet exemple, le prénom est un attribut multivalué, ici une personne peut avoir de 1 à 3 prénoms.

Conseil : Composition et attribut multivalué

Une composition avec une classe partie dotée d'un seul attribut peut s'écrire avec un attribut multivalué.

Complément : Voir aussi

Composition

f) Attribut composé

Définition

Un attribut peut être composé (ou composite) joue le rôle d'un groupe d'attributs (par exemple une adresse peut être un attribut composé des attributs numéro, type de voie, nom de la voie). Cette notion renvoie à la notion de variable de type `Record` dans les langages de programmation classiques.

Remarque

Ce concept est hérité de la modélisation E-A.

Attention : On utilise pas les attributs dérivés en UML

En UML on préfère l'usage de compositions (ou de *dataType*) aux attributs composés.

- On utilisera des composition ou des *dataTypes* pour les attributs composés.
- On utilisera des compositions pour les attributs composés et multivalués.

Complément : Voir aussi

Composition

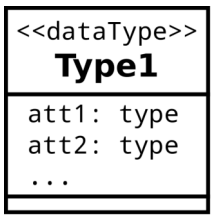
DataType

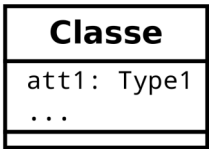
g) Stéréotype datatype

Définition : Datatype

Les *datatypes* permettent de définir des types complexes propres en extension des types primaires (entier, chaîne, date...).

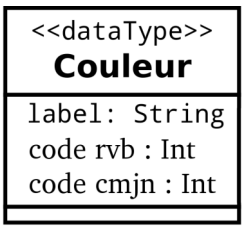
Syntaxe

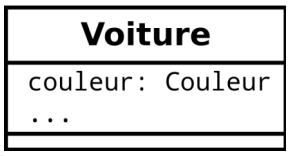




Stéréotype datatype

Exemple





Stéréotype datatype (exemple)

Méthode : Attributs composés

Les *datatypes* sont utilisés en base de données pour exprimer des attributs composés.

Cette modélisation est équivalente à une composition 1:1.

Complément

Stéréotype

2. Transformation de l'imbrication en relationnel

Objectifs

Savoir faire le passage d'un schéma conceptuel UML à un schéma relationnel dans tous les cas.

Reconnaître les cas de transformation qui se traitent toujours de la même façon et ceux qui nécessite une modélisation complémentaire.

a) Transformation des compositions

Clé locale

Pour identifier une classe partie dans une composition, on utilise une clé locale concaténée à la clé étrangère vers la classe composite, afin d'exprimer la dépendance entre les deux classes.

Méthode

Une composition

- est transformée comme une association 1:N,
- puis on combine une clé locale de la classe partie et la clé étrangère vers la classe composite pour construire une clé primaire composée.

Graphique 16 Composition

```
Classe1(#a,b)
Classe2(#c,#a=>Classe1,d)
```

Remarque : Clé candidate

Si une clé candidate (globale) permet d'identifier de façon unique une partie indépendamment du tout, on préférera la conserver comme clé candidate plutôt que de la prendre pour clé primaire.

Si on la choisit comme clé primaire cela revient à avoir transformé la composition en agrégation, en redonnant une vie propre aux objets composants.

Complément : Composition et entités faibles en E-A

Une composition est transformée selon les mêmes principes qu'une entité faible en E-A.

Rappel : Voir aussi

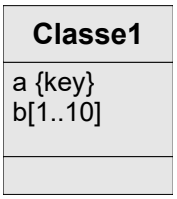
Transformation des attributs

b) Transformation des attributs multivalués

Méthode

Pour chaque attribut multivalué b d'une classe C,

- on crée une nouvelle relation RB,
- qui comprend un attribut monovalué correspondant à b,
- plus la clé de la relation représentant C ;
- la clé de RB est la concaténation des deux attributs.

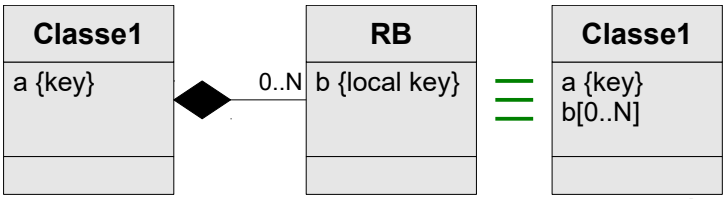


Graphique 17 Attribut multivalué

```
R1(#a)
RB(#b,#a=>Classe1)
```

Remarque

La transformation d'une composition avec un seul attribut pour la classe composante donne un résultat équivalent à la transformation d'un attribut multivalué.



Graphique 18 Composition et attribut multivalué

```
R1(#a)
RB(#b,#a=>Classe1)
```

Complément : Méthode alternative

Dans le cas où le nombre maximum de `b` est fini, et petit, on peut également adopter la transformation suivante : `R1(#a,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10)`.

Si le nombre d'attributs est infini (`b[1..*]`) c'est impossible, s'il est trop grand ce n'est pas souhaitable.

Rappel : Voir aussi

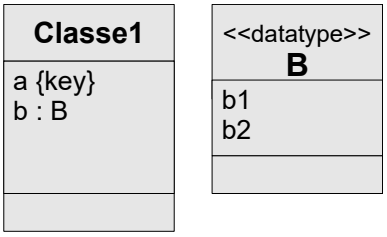
Transformation des attributs

c) Transformation des datatypes

Méthode : Datatype

Pour chaque datatype comprenant N sous-attributs,

- on crée N attributs correspondants,
- dont les noms sont la concaténation du nom de l'attribut composite avec celui du sous-attribut.

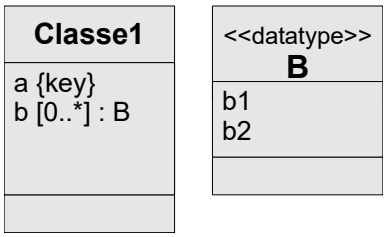


Graphique 19 Attribut composé

```
R1(#a,b_b1,b_b2)
```

Méthode : Datatype multivalué

On combine les règles énoncées pour les attributs composés et pour les attributs multivalués.



Graphique 20 Attribut composé multivalué

```
R1(#a)
RB(#b_b1,#b_b2,#a=>Classe1)
```

Remarque

La transformation des datatype multivalué est similaire à la transformation des compositions.

Rappel : Voir aussi

Transformation des compositions

d) Exercice : Pays

Soit le schéma UML suivant (on ignore les domaines à des fins de simplification) :

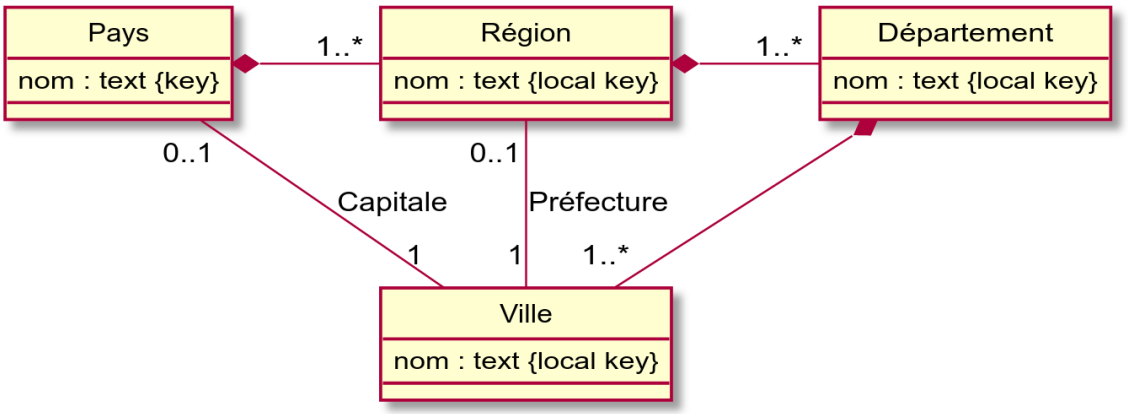


Image 20

Quels sont les modèles relationnels qui correspondent à ce modèle conceptuel ?

☐	Pays(#Nom, Capitale=>Ville) Region(#Nom, Prefecture=>Ville, Pays=>Pays) Departement(#Nom, Region=>Region) Ville(#Nom, Departement=>Departement)
☐	Pays(#Nom, Capitale=>Ville) Region(#Nom, Prefecture=>Ville, Pays=>Pays) Departement(#Nom, Region=>Region, Pays=>Pays) Ville(#Nom, Departement=>Departement, Region=>Region, Pays=>Pays)
☐	Pays(#Nom) Region(#Nom, Pays=>Pays) Departement(#Nom, Region=>Region) Ville(#Nom, Departement=>Departement, Capitale=>Pays, Prefecture=>Region)

☐	Pays(#Nom, Capitale=>Ville)
	Region(#Nom, #Pays=>Pays, Prefecture=>Ville)
	Departement(#Nom, #Region=>Region)
	Ville(#Nom, #Departement=>Departement)

☐	Pays(#Nom, CapitaleVille=>Ville, CapitaleDepartement=>Ville, CapitaleRegion=>Ville, CapitalePays=>Ville)
	Region(#Nom, #Pays=>Pays, PrefectureVille=>Ville, PrefectureDepartement=>Ville, PrefectureRegion=>Ville, PrefecturePays=>Ville)
	Departement(#Nom, #Region=>Region, #Pays=>Region)
	Ville(#Nom, #Departement=>Departement, #Region=>Departement, #Pays=>Departement)

e) Trousseau de clés

Attention

En UML et en relationnel, il existe plusieurs termes mobilisant le mot « clé », le seul concept qui est commun est le concept de **clé**. Tous les autres sont spécifiques au niveau conceptuel ou relationnel.

Rappel : Concept commun au niveau UML et relationnel

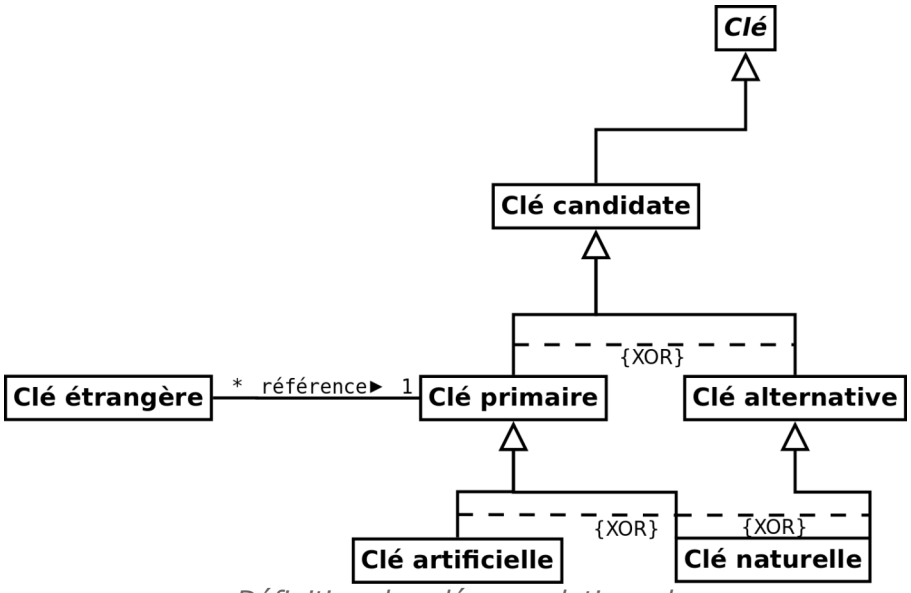
Clé (key)

Rappel : Concept spécifique au niveau UML

Clé locale (local key)

Rappel : Concepts spécifiques au niveau relationnel

Clé (key)
Clé candidate (candidate key)
Clé primaire (primary key) et Clé alternative (alternate key)
Clé artificielle (surrogate key) et Clé naturelle (natural key, business key)
Clé étrangère (foreign key)



Définition des clés en relationnel

B. Exercices

1. Objets Numériques Libres

[30 min]

L'association ONL (Objets Numériques Libres) est une association de promotion des logiciels libres. Elle souhaite exposer sur un site Internet une liste de logiciels libres. Ce site sera adossé à une base de données relationnelle ou relationnel-objet. La première étape de sa démarche est de réaliser un modèle conceptuel représentant ce qu'elle souhaite faire.

- La base de données permet de gérer des applications.

  Les applications sont identifiées par leur nom (LibreOffice, Gimp...) et leur version (1.0, 2.1), et comportent une description courte et une URL. Tous les attributs sont obligatoires. Chaque application a une URL unique.
- La base de données permet de gérer des librairies.

  Les librairies sont des logiciels mais pas des applications. Elles ont les mêmes attributs que les applications (nom, version, description courte, URL), mais les URL ne sont pas nécessairement uniques. Les applications peuvent dépendre de librairies ou d'autres applications, et les librairies peuvent dépendre d'autres librairies (mais pas d'une application).
- La base de données permet de gérer des composants.

  Les composants sont intégrés à une application ou librairie. Les composants ont un code interne à l'application ou la librairie qu'il servent, une version et une description courte, et une URL. Le code et le numéro de version permettent d'identifier localement le composant au sein de la librairie ou de l'application, la description courte et l'URL sont optionnelles.
- La base de données permet de gérer des licences.

  Les applications, librairies et composants sont attachés à une ou plusieurs licences identifiées par leur nom (GPL, MPL...), leur version et leur langue, et comportant le texte intégral de la licence. Les versions des logiciels et

licences sont de type "numéro de licence majeur point numéro de licence mineur", comme "1.0" ou "2.2".

- La base de données permet de gérer des catégories.

  Chaque logiciel est rangé dans une catégorie principale, et plusieurs catégories secondaires. Exemple de catégories : bureautique, dessin, multimédia, physique...

Exemple (factice) de données

- Applications :
  - Scenari 4.1, une chaîne éditoriale XML, http://scenari.org, dépend de Libreoffice 4.3 et de ImageMagick 6.8
  - Libreoffice 4.3, une suite bureautique WYSIWYG, http://libreoffice.org
- Librairie :
  - ImageMagick 6.8, permet de créer, de convertir, de modifier et d'afficher des images, http:// imagemagick.org
- Composant :
  - impng 0.2 est un composant de ImageMagick 6.8, permet de compresser une image au format PNG.
- Toutes ces applications, librairies et composants sont disponibles sous une licence LGPL 3.0 et GPL 3.0 françaises.
- Toutes ces applications et librairies sont rangées dans la catégorie principale "document". Scenari est rangé dans la catégorie secondaire "Édition WYSIWYM", Libreoffice dans la catégorie secondaire "Bureautique", ImageMagick dans la catégorie secondaire "Multimédia".

Question

Réaliser un MCD en UML.

2. Lab III

[30 min]

Description du problème

Un laboratoire souhaite gérer les médicaments qu'il conçoit.

- Un médicament est décrit par un nom, qui permet de l'identifier. En effet il n'existe pas deux médicaments avec le même nom. Un médicament comporte une description courte en français, ainsi qu'une description longue en latin. On gère aussi le conditionnement du médicament, c'est à dire le nombre de pilules par boîte (qui est un nombre entier).
- À chaque médicament on associe une liste dédiée de contre-indications, généralement plusieurs, parfois aucune. Les contre-indications sont triées par ordre d'importance. L'ordre est total est strict pour un médicament, donc chaque contre-indication possède une importance et il n'existe pas deux contre-indications associées au même médicament avec la même importance.
- Tout médicament possède au moins un composant, souvent plusieurs. Un composant est identifié par un code unique et possède un intitulé. Tout composant peut intervenir dans la fabrication de plusieurs médicaments. Il existe des composants qui ne sont pas utilisés pour fabriquer des médicaments et que l'on veut quand même gérer.

Données de test

Afin de matérialiser notre base de données, nous obtenons les descriptions suivantes :

- Le **Chourix** a pour description courte *« Médicament contre la chute des choux »* et pour description longue *« Vivamus fermentum semper porta. Nunc diam velit, adipiscing ut tristique vitae, sagittis vel odio. Maecenas convallis ullamcorper ultricies. Curabitur ornare. »*. Il est conditionné en boîte de 13.

  Ses contre-indications sont :
  a. Le Chourix ne doit jamais être pris après minuit.
  b. Le Chourix ne doit jamais être mis au contact avec de l'eau.

  Ses composants sont le **HG79** et le **SN50**.
- Le **Tropas** a pour description courte *« Médicament contre les dysfonctionnements intellectuels »* et pour description longue *« Suspendisse lectus leo, consectetur in tempor sit amet, placerat quis neque. Etiam luctus porttitor lorem, sed suscipit est rutrum non. »*. Il est conditionné en boîte de 42.

  Ses contre-indications sont :
  a. Le Tropas doit être gardé à l'abri de la lumière du soleil

  Son unique composant est le **HG79**.
- Les composants existants sont :
  - **HG79** : "Vif-argent allégé"
  - **HG81** : "Vif-argent alourdi"
  - **SN50** : "Pur étain"

Question 1

Effectuez le modèle conceptuel en UML de ce problème.

Indices :

> *On note dans l'énoncé que les contre-indications sont **dédiée** aux médicaments, et par ailleurs on note dans les données exemples que les contre-indications sont énoncées spécifiquement par rapport aux médicaments.*
>
> *Ce n'est pas parce que les composants s'appellent ainsi dans l'énoncé que l'on est en présence d'une composition.*

Question 2

En mobilisant les règles adéquates, proposer un modèle logique de données correspondant en relationnel. Le repérage des domaines et des clés est obligatoire.

Question 3

Dessiner des tableaux remplis avec les données fournies en exemple, afin de montrer que le modèle fonctionne selon le besoin exprimé initialement. On pourra mettre le premier mot seulement des descriptions pour gagner du temps.

3. Questionnaires

[45 min]

On souhaite construire une base de données de questions pour réaliser une application permettant à des utilisateurs de tester leurs connaissances, sur divers sujets.

On renseigne pour chaque question un niveau de difficulté indiqué par un entier entre 1 et 5, ainsi que la date de création de la question. Chaque question est dotée d'un numéro unique pour toute la base. De plus chaque question est unique,

il n'existe pas deux questions avec le même intitulé : par exemple il n'est pas possible d'avoir deux fois la question : « quelle est la capitale de la France ».

Il existe deux types de questions :

- Reconnaître : l'utilisateur doit reconnaître un élément sur une image. Il faut donc stocker dans la base de données un chemin vers l'image à afficher et la réponse attendue.

  Exemple d'intitulé : « Sur le diagramme UML ci-joint, donnez les noms des éléments qui sont des classes associations » (titre du thème : conception, intitulé de la catégorie : classes associations, difficulté 2).
- Vrai/Faux : l'utilisateur doit dire si l'intitulé de la question est vrai ou faux. On doit enregistrer la réponse attendue (vrai ou faux).

  Exemple d'intitulé : « Dans un schéma relationnel, on fait apparaître les attributs dérivés » (titre du thème : schéma relationnel, intitulé de la catégorie : traduction UML/relationnel, difficulté 2).

Un thème possède un titre et un domaine. Un thème est composé de plusieurs catégories (on utilisera une composition). Chaque catégorie a un intitulé. Il existe plusieurs catégories avec un même intitulé, mais jamais à l'intérieur d'un même thème.

Toutes les questions relèvent d'une unique catégorie.

Chaque fois qu'un utilisateur répond à une question, on enregistre le résultat (bonne ou mauvaise réponse).

On souhaite également conserver des informations sur les utilisateurs : son login (qui est une clé), son nom et son prénom. Par ailleurs, on veut calculer le score d'un utilisateur : chaque question à laquelle il a répondu correctement rapporte autant de points que le niveau de difficulté.

Question 1

Proposez un modèle conceptuel en UML répondant à ces besoins.

Vous veillerez à expliciter les hypothèses faites et à faire apparaître les clés candidates et les autres contraintes sur le schéma.

Question 2

Proposez un modèle relationnel à partir du schéma UML. Vous justifierez les choix non triviaux. Vous proposerez un modèle en 3NF. Vous exprimerez les contraintes issues de la modélisation UML.

Question 3

Écrivez le code SQL LDD permettant la création de la table Catégorie.

Question 4

Dessinez tous les tableaux permettant de représenter les deux questions présentées en exemple dans l'énoncé. Remplissez avec une information pertinente à votre convenance les champs pour lesquels vous n'avez pas d'information.

Analyse de bases de données SQL avec les agrégats (GROUP BY)

VI

A. Cours

1. Introduction aux agrégats avec GROUP BY

Objectifs

Savoir réaliser un agrégat en SQL en utilisant les fonctions de regroupement et les clause GROUP BY

a) Exercice : La somme finale

Que renvoie la dernière instruction SQL de la séquence ci-dessous ?

```
1  CREATE TABLE R1 (a INTEGER, b INTEGER);
2  CREATE TABLE R2 (b INTEGER);
3  INSERT INTO R1 VALUES (1,1);
4  INSERT INTO R1 VALUES (2,1);
5  INSERT INTO R2 VALUES (1);
6  INSERT INTO R2 VALUES (2);
7  SELECT sum(R1.a) FROM R1, R2 WHERE R1.b=R2.b;
```

b) Définition de l'agrégation

Définition : Agrégat

Un agrégat est un partitionnement horizontal d'une table en sous-tables, en fonction des valeurs d'un ou plusieurs attributs de partitionnement, suivi éventuellement de l'application d'une fonction de calcul à chaque attribut des sous-tables obtenues.

Synonyme : Regroupement

Syntaxe

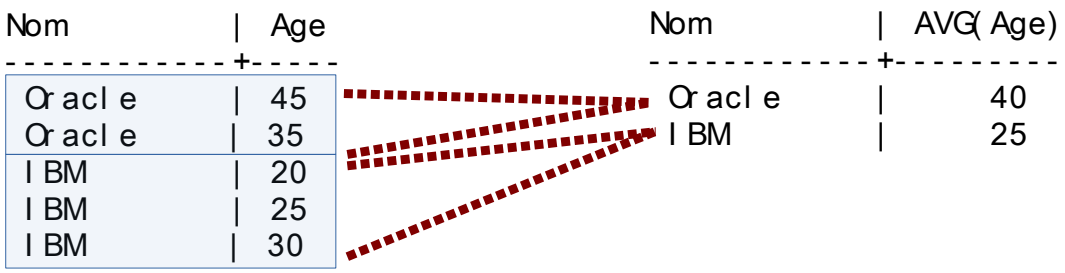
```
1  SELECT liste d'attributs de partionnement à projeter et de fonctions
   de calcul
2  FROM liste de relations
3  WHERE condition à appliquer avant calcul de l'agrégat
4  GROUP BY liste ordonnée d'attributs de partitionnement
```

1. La table est divisée en sous-ensembles de lignes, avec un sous-ensemble pour chaque valeur différente des attributs de partitionnement projetés dans le SELECT
2. Les fonctions d'agrégation sont appliquées sur les attributs concernés

Exemple

```
1  SELECT Societe.Nom, AVG(Personne.Age)
2  FROM Personne, Societe
3  WHERE Personne.NomSoc = Societe.Nom
4  GROUP BY Societe.Nom
```

Societe.Nom est un ici le seul attribut de partitionnement, donc un sous ensemble est créé pour chaque valeur différente de *Societe.Nom*, puis la moyenne (fonction AVG) est effectuée pour chaque sous-ensemble.



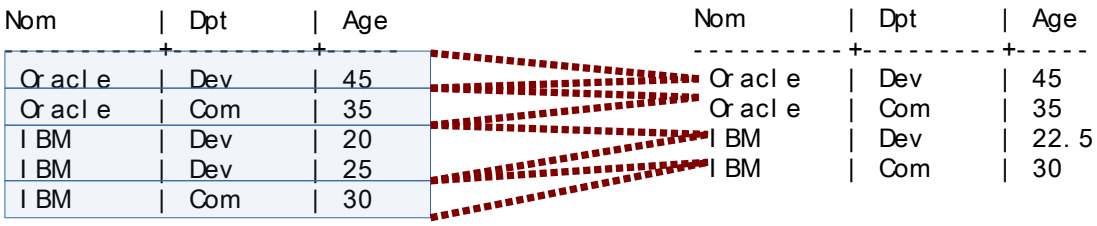
Regroupement avec un attribut et une fonction

Cette requête calcul l'âge moyen du personnel pour chaque société.

Exemple

```
1  SELECT Societe.Nom, Societe.Dpt AVG(Personne.Age)
2  FROM Personne, Societe
3  WHERE Personne.NomSoc = Societe.Nom
4  GROUP BY Societe.Nom, Societe.Dpt
```

Societe.Nom et *Societe.Dpt* sont les deux attributs de partitionnement, donc un sous-ensemble est créé pour chaque valeur différente du couple (*Societe.Nom*,*Societe.Dpt*).



Regroupement avec deux attributs et une fonction

Cette requête calcul l'âge moyen du personnel pour chaque département de chaque société.

Remarque : Requête inutile

La requête est inutile dès lors que l'agrégat est défini sur une valeur unique dans la relation, puisqu'on aura bien une ligne par ligne de la table source.

```
1  SELECT Societe.Nom
2  FROM Societe
3  GROUP BY Societe.Nom
```

Exemple d'agrégat inutile (countrycode est une clé de country)

c) Exemple d'attribut d'agrégation

Schéma relationnel

```
1  country(#countrycode:char(2), name:varchar, population:numeric)
2  city(#code:char(3), countrycode=>country, name:varchar,
   population:numeric):
```

Schéma de base de données

```
1  CREATE TABLE country (
2  countrycode CHAR(2) NOT NULL,
3  name VARCHAR NOT NULL,
4  population NUMERIC(3),
5  PRIMARY KEY (countrycode)
6  );
7
8  CREATE TABLE city (
9  citycode CHAR(3) NOT NULL,
10 countrycode CHAR(2) NOT NULL,
11 name VARCHAR NOT NULL,
12 population NUMERIC(2,1),
13 PRIMARY KEY (citycode),
14 FOREIGN KEY (countrycode) REFERENCES country(countrycode)
15 );
```

Données

```
1  INSERT INTO country VALUES ('ES', 'Spain', 46);
2  INSERT INTO country VALUES ('FR', 'France', 67);
```

```
3  INSERT INTO country VALUES ('DE', 'Germany', 82);
4
5  INSERT INTO city VALUES ('BAR', 'ES', 'Barcelona', 1.9);
6  INSERT INTO city VALUES ('MAD', 'ES', 'Madrid', 3.3);
7  INSERT INTO city VALUES ('ZAR', 'ES', 'Zaragoza', 0.7);
8
9  INSERT INTO city VALUES ('PAR', 'FR', 'Paris', 2.2);
10 INSERT INTO city VALUES ('LYO', 'FR', 'Paris', 0.5);
11 INSERT INTO city VALUES ('LLL', 'FR', 'Lille', 0.2);
12 INSERT INTO city VALUES ('AMN', 'FR', 'Amiens', 0.1);
```

Sélectionne les countrycode existants dans la table city

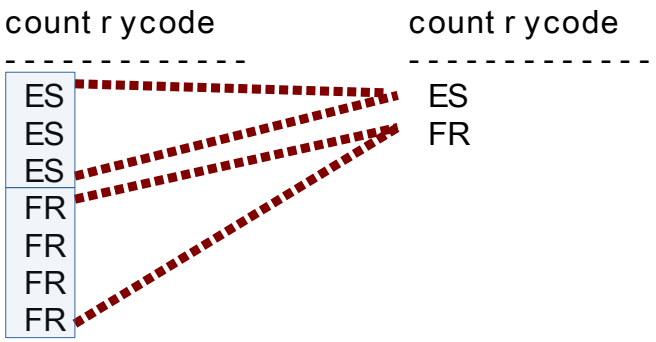
```
1  SELECT countrycode
2  FROM city;
```

```
1   countrycode
2  -------------
3   ES
4   ES
5   ES
6   FR
7   FR
8   FR
9   FR
```

Sélectionne les countrycode existants dans la table city avec agrégat

```
1  SELECT countrycode
2  FROM city
3  GROUP BY countrycode;
```

```
1   countrycode
2  -------------
3   FR
4   ES
```



Principe de l'agrégation

d) Fonctions d'agrégation

Définition : Fonctions d'agrégation

Une fonction d'agrégation (ou fonction de regroupement) s'applique aux valeurs du sous-ensemble d'un agrégat e relation avec pour résultat la production d'une valeur atomique unique (entier, chaîne, date, etc).

Les cinq fonctions prédéfinies sont :

- **COUNT (Relation.Propriété)**
  Renvoie le nombre de valeurs non nulles d'une propriété pour tous les tuples d'une relation ;
- **SUM (Relation.Propriété)**
  Renvoie la somme des valeurs d'une propriété des tuples (numériques) d'une relation ;
- **AVG (Relation.Propriété)**
  Renvoie la moyenne des valeurs d'une propriété des tuples (numériques) d'une relation ;
- **MIN (Relation.Propriété)**
  Renvoie la plus petite valeur d'une propriété parmi les tuples d'une relation .
- **MAX (Relation.Propriété)**
  Renvoie la plus grande valeur d'une propriété parmi les tuples d'une relation.

Attention : Fonctions de calcul sans partitionnement

Si une ou plusieurs fonctions de calcul sont appliquées sans partitionnement, le résultat de la requête est un tuple unique.

Exemple

```
1 SELECT Min(Age), Max(Age), Avg(Age)
2 FROM Personne
3 WHERE Qualification='Ingénieur'
```

Remarque : Comptage d'une relation

Pour effectuer un comptage sur tous les tuples d'une relation, appliquer la fonction `count` à un attribut de la clé primaire. En effet cet attribut étant non nul par définition, il assure que tous les tuples seront comptés.

e) Exemple de fonctions d'agrégation

Rappel : Schéma relationnel

```
1 country(#countrycode:char(2), name:varchar, population:numeric)
2 city(#code:char(3), countrycode=>country, name:varchar,
  population:numeric):
```

Exemple d'attribut d'agrégation

i Agrégat avec un attribut d'agrégation et une fonction d'agrégation

Requête sans agrégat

Sélectionne les *countrycode* et les *citycode* existants dans la table *city*.

```
1 SELECT countrycode, citycode
2 FROM city;
```

```
1  countrycode | citycode
```

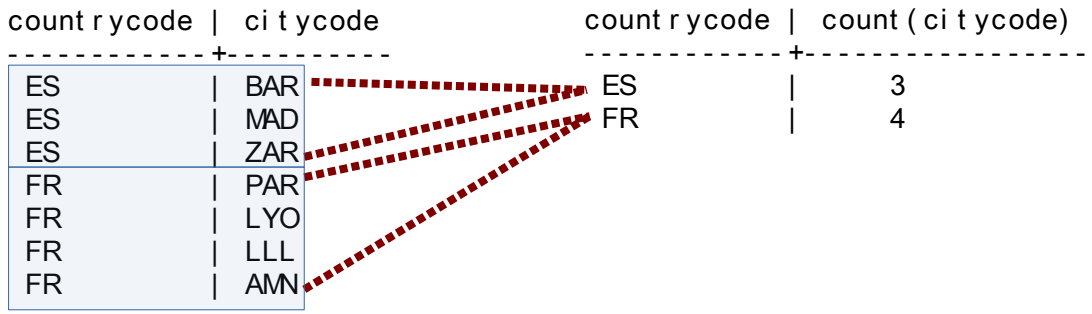
```
2  -------------+----------
3   ES          | BAR
4   ES          | MAD
5   ES          | ZAR
6   FR          | PAR
7   FR          | LYO
8   FR          | LLL
9   FR          | AMN
```

Requête avec agrégat

- Sélectionne les *countrycode* et les *citycode* existants dans la table *city*,
- puis agrège par valeurs distinctes de *countrycode*.

```
1  SELECT countrycode, count(citycode)
2  FROM city
3  GROUP BY countrycode;
```

```
1   countrycode | count
2  -------------+-------
3   FR          |     4
4   ES          |     3
```



Regroupement avec fonction d'agrégation

ii Agrégat avec un attribut d'agrégation et deux fonctions d'agrégation

Sélectionne les *countrycode*, les *citycode* et les *populations* existants dans la table *city*.

```
1  SELECT countrycode, citycode, population
2  FROM city;
```

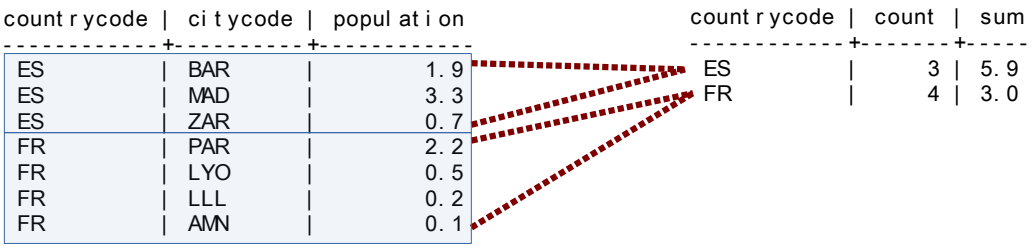
```
1   countrycode | citycode | population
2  -------------+----------+------------
3   ES          | BAR      |        1.9
4   ES          | MAD      |        3.3
5   ES          | ZAR      |        0.7
6   FR          | PAR      |        2.2
7   FR          | LYO      |        0.5
8   FR          | LLL      |        0.2
9   FR          | AMN      |        0.1
```

Requête avec agrégat

- Sélectionne les *countrycode*, les *citycode* et les *populations* existants dans la table *city*,
- puis agrège par valeurs distinctes de *countrycode*
- puis calcule les fonctions *count* et *sum*.

```
SELECT countrycode, count(citycode), sum(population)
FROM city
GROUP BY countrycode;
```

```
 countrycode | count | sum
-------------+-------+-----
 FR          |     4 | 3.0
 ES          |     3 | 5.9
```



Regroupement avec deux fonctions d'agrégation

iii Agrégat sans attribut d'agrégation et avec une fonction d'agrégation

Requête sans agrégat

Sélectionne les *populations* existants dans la table *city*.

```
SELECT population
FROM city;
```

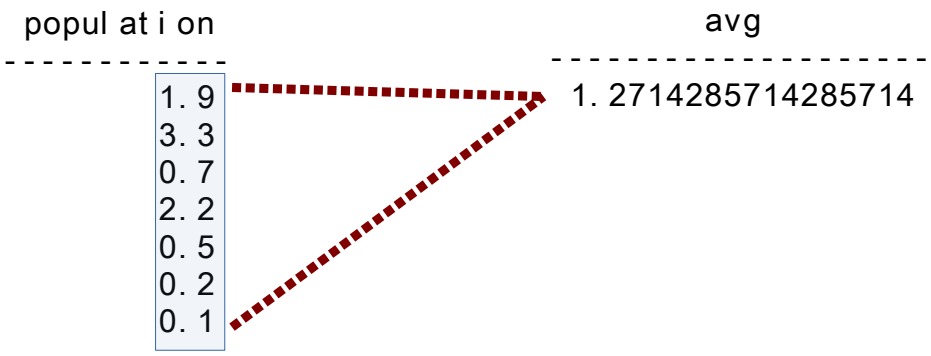
```
 population
------------
        1.9
        3.3
        0.7
        2.2
        0.5
        0.2
        0.1
```

Requête avec agrégat

- Sélectionne les *populations* existants dans la table *city*,
- puis calcule la fonction *avg* (pour *average*, moyenne).

```
SELECT avg(population)
FROM city;
```

```
        avg
--------------------
 1.2714285714285714
```



Regroupement avec fonction d'agrégation sans GROUP BY

2. Approfondissement des agrégats avec GROUP BY et HAVING

Objectifs

Savoir réaliser un agrégat en SQL en utilisant les fonctions de regroupement et les clause GROUP BY et HAVING

a) Single-Value Rule

Fondamental : Principe de la "Single-Value Rule" établie par le standard SQL

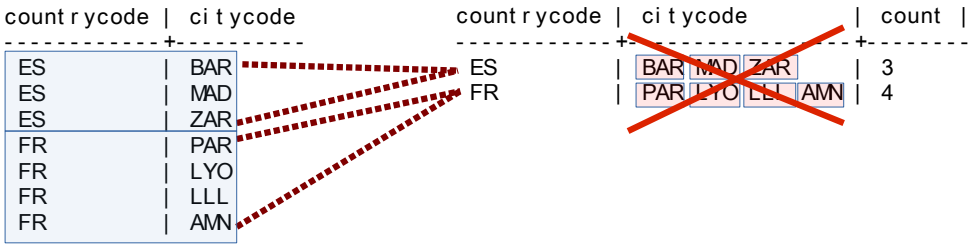
Toute colonne de la clause SELECT doit soit :

- être un attribut d'agrégation (et être également présente dans la clause GROUP BY) ;
- être attribut de calcul (présent dans une fonction d'agrégation.)

Attention : Requête illégale

```
1   SELECT countrycode, citycode, COUNT(citycode)
2   FROM city
3   GROUP BY countrycode;
```

```
1   ERROR:  column "city.citycode" must appear in the GROUP BY clause or
    be used in an aggregate function;
```



Tentative de GROUP BY illégal

La requête est non standard et non logique car on cherche à mettre plusieurs données dans la case *citycode* après le GROUP BY (il y a plusieurs *citycode* par

countrycode).
Elle sera refusée par tous les SGBD.

Complément : Single-Value Rule

https://mariadb.com/kb/en/sql-99/the-single-value-rule[1]

i Tolérance (de certains SGBD)

Requête non standard tolérée par certains SGBD

```
1  SELECT citycode, countrycode, AVG(population)
2  FROM city
3  GROUP BY citycode;
```

```
1  citycode | countrycode |          avg
2 ----------+-------------+------------------------
3  LLL      | FR          | 0.20000000000000000000
4  BAR      | ES          | 1.90000000000000000000
5  PAR      | FR          |     2.2000000000000000
6  LYO      | FR          | 0.50000000000000000000
7  ZAR      | ES          | 0.70000000000000000000
8  AMN      | FR          | 0.10000000000000000000
9  MAD      | ES          |     3.3000000000000000
```

La requête est non standard, même si c'est logiquement acceptable, car il n'y a qu'un *countrycode* par *citycode* ici, *city* étant une clé.

Certains SGBD comme Postgres acceptent donc cette syntaxe, en ajoutant implicitement l'attribut qui manque au GROUP BY, car il n'y a pas d'ambiguïté si l'on analyse le graphe des DF : *citycode* → *countrycode*. Mais le standard impose d'être explicite, car le SGBD a le droit de ne pas le deviner.

ii Tolérance partielle

Requête légale

```
1  SELECT ci.countrycode, co.name, count(*)
2  FROM city ci JOIN country co
3  ON ci.countrycode=co.countrycode
4  GROUP BY ci.countrycode, co.name;
```

```
1  countrycode |  name   | count
2 -------------+---------+-------
3  FR          | France  |     4
4  ES          | Spain   |     3
```

Requête non standard tolérée par certains SGBD

```
1  SELECT co.countrycode, co.name, count(*)
2  FROM city ci JOIN country co
3  ON ci.countrycode=co.countrycode
4  GROUP BY co.countrycode;
```

La requête est non standard, car *co.name* est dans le SELECT mais pas dans le GROUP BY. Comme dans l'exemple précédent certains SGBD accepteront cette requête grâce à la DF évidente : *co.countrycode* → *co.name*

1 - https://mariadb.com/kb/en/sql-99/the-single-value-rule/

Attention : Requête non standard non tolérée

Mais si l'on utilise à présent ***ci.countrycode*** à la place de *co.countrycode*...

```
1  SELECT ci.countrycode, co.name, count(*)
2  FROM city ci JOIN country co
3  ON ci.countrycode=co.countrycode
4  GROUP BY ci.countrycode;
```

```
1  ERROR:  column "co.name" must appear in the GROUP BY clause or be
   used in an aggregate function
```

La requête est non standard, *co.name* doit être mentionné dans la clause GROUP BY. Logiquement on pourrait juger cela superflu, car par transitivité *ci.countrycode* → *co.name*, mais le SGBD (ici Postgres) ne va pas jusque là.

Conseil

Il est donc conseillé d'appliquer le *Single-Value Rule* et de toujours expliciter tous les attributs dans le GROUP BY, pour éviter la dépendance au SGBD ou bien les erreurs liées à des variations mineures de syntaxe.

b) Approfondissement de la fonction COUNT

Syntaxe : COUNT(*)

- `COUNT(*)` renvoie le nombre de lignes de la relation.
- `COUNT(a)` renvoie le nombre de valeurs **non nulles** de la relation (donc `COUNT(*)` et `COUNT(a)` sont équivalents si a est non null)
- `COUNT (DISTINCT a)` renvoie le nombre de valeurs **non nulles** et **distinctes** de la relation

Exemple

```
1  CREATE TABLE t (
2  pk INTEGER PRIMARY KEY,
3  a VARCHAR(1),
4  b VARCHAR(1));
5  INSERT INTO t VALUES (1, 'a', 'x');
6  INSERT INTO t VALUES (2, 'a', 'y');
7  INSERT INTO t VALUES (3, 'b', NULL);
```

#pk	a NOT NULL	b
1	a	x
2	a	y
3	b	

Tableau 48 table t

- `SELECT COUNT(*) FROM t` : 3
- ≡ `SELECT COUNT(pk) FROM t` : 3
- ≡ `SELECT COUNT(a) FROM t` : 3
- `SELECT COUNT (DISTINCT a) FROM t` : 2
- ≡ `SELECT COUNT(*) FROM t GROUP BY a` : 2

Attention

Pour des raisons de performance, on préfère l'usage d'un `GROUP BY` à `COUNT (DISTINCT a)`. On utilise ce dernier que s'il n'existe pas de solution ne mobilisant qu'une seule requête.

Exemple

```
1  SELECT COUNT(*) AS nb_total, COUNT (DISTINCT a) AS nb_a FROM t;
```

```
1   nb_total | nb_a
2  ----------+------
3          3 |    2
```

Attention : COUNT(*) et résultat de requête

Il faut prendre garde à l'usage de `COUNT(*)` dès que l'on adresse des requêtes portant sur plusieurs tables, en effet cela comptera **toutes** les lignes produite par le produit cartésien, ce qui n'est pas toujours ce que l'on souhaite.

On préférera compter un attribut non nul.

Exemple : COUNT(*) et jointure externe

Soit la séquence suivante permettant d'instancier t1 et t2, tel que référence t1 référence t2 et qu'il existe un élément de t2 qui n'est jamais référencé.

```
1  CREATE TABLE t2 (
2  pk INTEGER PRIMARY KEY,
3  a VARCHAR
4  );
5
6  CREATE TABLE t1 (
7  pk VARCHAR PRIMARY KEY,
8  fk INTEGER REFERENCES t2(pk)
9  );
10
11 INSERT INTO t2 VALUES (1, 'x');
12 INSERT INTO t2 VALUES (2, 'x');
13
14 INSERT INTO t1 VALUES ('a',1);
15 INSERT INTO t1 VALUES ('b',1);
```

Si l'on utilise un `COUNT(*)` alors on obtient la valeur 1 pour `b` au lieu du 0 escompté :

- la ligne `(b,NULL)` est ajoutée au résultat parce que `b` n'est jamais référencé (principe de la jointure externe)
- étant donné que la jointure externe a lieu **avant** le regroupement et que `COUNT(*)` compte toutes les lignes

```
1  SELECT t2.pk, COUNT(*)
2  FROM t2
3  LEFT JOIN t1 ON t1.fk=t2.pk
4  GROUP BY t2.pk;
```

```
1   pk | count
2  ----+-------
3    1 |     2
```

```
4    2 |     1
```

Si l'on utilise un `COUNT(a)` sur la clé de la table t1, celle-ci étant nulle dans le cas des lignes ajoutées par la jointure externe, le fonctionnement est celui attendu.

```
1  SELECT t2.pk, COUNT(t1.pk)
2  FROM t2
3  LEFT JOIN t1 ON t1.fk=t2.pk
4  GROUP BY t2.pk;
```

```
1   pk | count
2  ----+-------
3    1 |     2
4    2 |     0
```

Complément

https://sql.sh/fonctions/agregation/count[2]

c) Restriction après agrégation (HAVING)

Définition

La clause HAVING permet d'effectuer une second restriction après l'opération d'agrégation.

Syntaxe

```
1  SELECT liste d'attributs de partionnement à projeter et de fonctions
   de calcul
2  FROM liste de relations
3  WHERE condition à appliquer avant calcul de l'agrégat
4  GROUP BY liste ordonnée d'attributs de partitionnement
5  HAVING condition sur les fonctions de calcul
```

Exemple

```
1  SELECT Societe.Nom, AVG(Personne.Age)
2  FROM Personne, Societe
3  WHERE Personne.NomSoc = Societe.Nom
4  GROUP BY Societe.Nom
5  HAVING COUNT(Personne.NumSS) > 2
```

Cette requête calcul l'âge moyen du personnel pour chaque société comportant plus de 2 salariés.

2 - https://sql.sh/fonctions/agregation/count

d) Ordre de résolution des requêtes SQL

Fondamental

L'ordre de résolution standard d'une requête SQL est :

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Attention : Usage des alias dans le GROUP BY ou le HAVING : requête non standard, acceptée par certains SGBD

```
1  SELECT substring(name,1,1) AS initiale, count(citycode)
2  FROM city
3  GROUP BY initiale;
```

```
1   initiale | count
2  ----------+-------
3   L        |     1
4   Z        |     1
5   B        |     1
6   M        |     1
7   P        |     2
8   A        |     1
9
```

La requête n'est pas standard, les résolutions de l'alias et de la fonction *substring* sont dans le SELECT, qui est postérieur à la résolution du GROUP BY. Postgres acceptera néanmoins cette syntaxe, ce qui évite de créer une vue ou d'utiliser une sous-requête.

Remarque : Restriction

Une restriction peut être appliquée avant calcul de l'agrégat, au niveau de la clause WHERE, portant ainsi sur la relation de départ, mais aussi après calcul de l'agrégat sur les résultats de ce dernier, au niveau de la clause HAVING.

Remarque : Utilisation de fonctions d'agrégation

Les fonctions d'agrégation peuvent être utilisées dans la claude HAVING ou dans la clause ORDER BY. Les fonctions **ne peuvent pas** être utilisées dans la clause WHERE (qui est résolu avant le GROUP BY).

e) Re-représentation de représentants

[30 minutes]

Soit le schéma relationnel suivant :

```
1  REPRESENTANTS (#NR, NOMR, VILLE)
2  PRODUITS (#NP, NOMP, COUL, PDS)
3  CLIENTS (#NC, NOMC, VILLE)
4  VENTES (#NR=>REPRESENTANTS(NR), #NP=>PRODUITS(NP), #NC=>CLIENTS(NC),
   QT)
```

Écrire en SQL les requêtes permettant d'obtenir les informations suivantes.

```
1  /* Les requêtes peuvent être testées dans un SGBDR, en créant une
   base de données avec le script SQL suivant */
2
3  /*
4  DROP TABLE VENTES ;
5  DROP TABLE CLIENTS ;
6  DROP TABLE PRODUITS ;
7  DROP TABLE REPRESENTANTS ;
8  */
9
10 CREATE TABLE REPRESENTANTS (
11  NR INTEGER PRIMARY KEY,
12  NOMR VARCHAR,
13  VILLE VARCHAR
14 );
15
16 CREATE TABLE PRODUITS (
17  NP INTEGER PRIMARY KEY,
18  NOMP VARCHAR,
19  COUL VARCHAR,
20  PDS INTEGER
21 );
22
23 CREATE TABLE CLIENTS (
24  NC INTEGER PRIMARY KEY,
25  NOMC VARCHAR,
26  VILLE VARCHAR
27 );
28
29 CREATE TABLE VENTES (
30  NR INTEGER REFERENCES REPRESENTANTS(NR),
31  NP INTEGER REFERENCES PRODUITS(NP),
32  NC INTEGER REFERENCES CLIENTS(NC),
33  QT INTEGER,
34  PRIMARY KEY (NR, NP, NC)
35 );
36
37 INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (1, 'Stephane',
   'Lyon');
38 INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (2, 'Benjamin',
   'Paris');
39 INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (3, 'Leonard',
   'Lyon');
40 INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (4, 'Antoine',
   'Brest');
41 INSERT INTO REPRESENTANTS (NR, NOMR, VILLE) VALUES (5, 'Bruno',
   'Bayonne');
```

```
42
43 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (1, 'Aspirateur',
   'Rouge', 3546);
44 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (2, 'Trottinette',
   'Bleu', 1423);
45 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (3, 'Chaise',
   'Blanc', 3827);
46 INSERT INTO PRODUITS (NP, NOMP, COUL, PDS) VALUES (4, 'Tapis',
   'Rouge', 1423);
47
48 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (1, 'Alice', 'Lyon');
49 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (2, 'Bruno', 'Lyon');
50 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (3, 'Charles',
   'Compiègne');
51 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (4, 'Denis',
   'Montpellier');
52 INSERT INTO CLIENTS (NC, NOMC, VILLE) VALUES (5, 'Emile',
   'Strasbourg');
53
54 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 1, 1, 1);
55 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 1, 2, 1);
56 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (2, 2, 3, 1);
57 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (4, 3, 3, 200);
58 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 2, 190);
59 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (1, 3, 2, 300);
60 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 2, 120);
61 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 4, 120);
62 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 4, 2);
63 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 1, 3);
64 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 4, 1, 5);
65 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 3, 1);
66 INSERT INTO VENTES (NR, NP, NC, QT) VALUES (3, 1, 5, 1);
```

Question 1

Le nombre de clients.

Question 2

Le nombre de produits de couleur rouge.

Question 3

Le nombre de clients par ville.

Question 4

La quantité totale des produits rouge vendus par chaque représentant.

Question 5

La quantité totale de produits rouges vendus pour chaque représentant ayant vendu plus de 5 fois des produits rouges (ayant réalisé plus de 5 ventes différentes de produits rouges).

B. Exercices

1. Location d'appartements en groupe

[20 min]

Soit le schéma relationnel suivant gérant le fonctionnement d'une agence de location d'appartements.

```
1 APPARTEMENT(#code_appt:String, adresse:String, type:
  {studio,F1,F2,F3,F4,F5+}, prix_loyer:Real)
2 LOCATAIRE(#code_loc:String, nom:String, prenom:String)
3 LOCATION(#code_loc=>Locataire, #code_appt=>Appartement)
4 PAIEMENT_LOYER(#code_loc=>Locataire, #code_appt=>Appartement,
  #date_payement:Date, prix_paye:Real)
```

Question 1

En SQL afficher le nombre d'appartements de chaque type, uniquement pour les types qui commencent par la lettre F.

Question 2

En SQL afficher le total payé par locataire (avec son code, nom et prenom) pour l'ensemble de ses appartements.

Question 3

En SQL afficher les locataires (code uniquement) qui louent au moins 2 appartements, en précisant le nombre d'appartements loués et la moyenne des loyers, et trié par ordre décroissant de cette moyenne.

2. Championnat de Formule 1

[20 min]

La base de données suivante permet de gérer les résultats des courses de Formule 1 dans un championnat.

```
1 CHAMPIONNAT(#nom:string, annee:integer)
2 CIRCUIT(#nom:string, ville:string)
3 COURSE(#nom:string,circuit=>CIRCUIT(nom),
  championnat=>CHAMPIONNAT(nom))
4 SPONSOR(#nom:string)
5 ECURIE(#nom:string, devise:string, couleur:string,
  sponsor=>SPONSOR(nom))
6 PILOTE(#id:integer, nom:string, prenom:string, ecurie=>ECURIE(nom))
7 TOUR(#pilote => PILOTE(id), #course => COURSE(nom), #num:integer,
  duree:integer)
```

Question 1

En algèbre relationnel et en SQL, afficher la liste de tous les pilotes dont le nom commence par la lettre 'D'.

Question 2

En SQL, afficher le nombre de pilotes par écurie en classant les résultats par ordre alphabétique des noms des écuries.

Question 3

En algèbre relationnel et en SQL, afficher les noms des sponsors qui ne sont pas liés à une écurie.

Question 4

En SQL, afficher le numéro, nom, prénom et écurie, avec leur temps moyen par tour, des pilotes participant à la course `Daytonutc 500` ; mais en ne conservant que les pilotes qui ont effectués au moins deux tours de piste, et en classant le résultat par temps moyen décroissant.

3. Questions scolaires

[30 min]

Soit le schéma relationnel suivant gérant le fonctionnement d'une école et les notes des élèves.

```
1  CLASSE(#intitule)
2  MATIERE(#intitule)
3  ELEVE(#login, nom, prenom, age, classe=>CLASSE)
4  ENSEIGNANT(#login, nom, prenom, mail, matiere=>MATIERE)
5  ENSEIGNE(#enseignant=> ENSEIGNANT, #classe=>CLASSE)
6  NOTE(#eleve=>ELEVE, #enseignant=>ENSEIGNANT, #date_examen, note)
7
8  Contrainte : un enseignant ne peut mettre une note à un élève que si
   celui-ci se trouve dans une classe dans laquelle il enseigne.
```

NB : Nous n'utiliserons pas de sous-requêtes.

Question 1

En algèbre relationnel **et** en SQL, afficher la liste des tous les étudiants dont le nom commence par A.

Question 2

En algèbre relationnel **et** en SQL, afficher les noms et prénoms des enseignants qui n'enseignent à aucune classe.

Question 3

En SQL, affichez le nombre d'étudiants enregistrés en "Terminale S 2".

Question 4

En SQL, affichez les logins, noms, prénoms, classes et moyennes des élèves en cours de "Mathématiques", par ordre décroissant de moyenne, à condition qu'ils aient au minimum 2 notes dans cette matière.

Question 5

En SQL, à des fins de statistiques, nous souhaitons rechercher par enseignant et par classe, les classes qui n'ont pas la moyenne générale, et afficher pour celles-ci : le nom, prénom et mail de l'enseignant en question, la matière enseignée, la classe, la moyenne d'âge des étudiants avec les extrêmes (minimum et maximum), la moyenne générale de la classe avec les valeurs extrêmes, ainsi que le nombre d'étudiants présents dans cette classe ; le tout classé par ordre alphabétique de classe, puis de nom et de prénom de l'enseignant.

Indice :

> *On fera l'hypothèse que tous les étudiants d'une classe ont le même nombre de notes (pas d'absence aux examens).*

4. Quiz : SQL LMD

Exercice 1

Soit les deux relations suivantes UV1 et UV2 représentant respectivement les places disponibles dans les cours de l'UTC et les inscriptions effectives pour les cours ouverts ce semestre :

Nom	Nombre
NF17	168
NF18	48
NF19	48
NF20	48
NF21	48
NF22	168

UV1 : Place disponibles

Nom	Nombre
NF17	182
NF18	46
NF19	44
NF22	98

UV2 : Inscriptions

Tableau 49 Relations UV1 et UV2

*Compléter la requête SQL suivante pour qu'elle retourne pour **tous** les cours avec leur nom, plus le nombre d'inscrits en excès ou en défaut pour les cours ouverts et NULL pour les cours fermés.*

SELECT UV1.Nom,

FROM UV1 UV2

ON

Exercice 2

Soit les deux relations suivantes UV1 et UV2 représentant respectivement les places disponibles dans les cours de l'UTC et les inscriptions effectives pour les cours ouverts ce semestre :

Nom	Nombre
NF17	168
NF18	48
NF19	48
NF20	48
NF21	48
NF22	168

UV1 : Place disponibles

Nom	Nombre
NF17	182
NF18	46
NF19	44
NF22	98

UV2 : Inscriptions

Tableau 50 Relations UV1 et UV2

Compléter la requête SQL suivante pour qu'elle retourne, pour les cours ouverts uniquement, le nombre d'inscrits moyen selon le nombre de places (nombre d'inscrits moyens pour les cours de 168 places, pour ceux de 48 places, etc.)

SELECT UV1.Nombre,

FROM UV1 UV2

ON

Exercice 3

Que renvoie la dernière instruction SQL de la séquence ci-dessous ?

```
1  CREATE TABLE R (a INTEGER, b INTEGER);
2  INSERT INTO R VALUES (1,1);
3  INSERT INTO R VALUES (1,2);
4  INSERT INTO R VALUES (3,3);
```

```
5  SELECT sum(R1.b) FROM R R1, R R2 WHERE R1.a=R2.a;
```

Théorie de la normalisation relationnelle

VII

A. Cours

La théorie de la normalisation relationnelle est très importante pour la conception de BD, dans la mesure où elle donne le cadre théorique pour la gestion de la redondance, et dans la mesure où une bonne maîtrise de la redondance est un aspect majeur de cette conception.

1. Redondance et normalisation

Objectifs

Comprendre la problématique de la redondance.

a) Introduction à la redondance

Soit la relation R suivante, définie en extension :

A	B	C	D	E	F	G
0	1	1	10	5	X	A
0	2	1	10	9	X	G
0	1	2	10	6	X	S
0	1	3	10	7	X	D
1	2	3	20	7	Y	D
0	3	3	10	9	X	G
1	4	3	20	8	Y	F
1	1	4	20	9	Y	G

Tableau 51 Relation R

Question 1

Proposez des clés pour cette relation. Justifiez brièvement.

Question 2

Cette relation contient-elle des redondances ? Si oui lesquelles ? Justifiez brièvement.

Question 3

Si la relation contient des redondances, proposez une solution contenant exactement la même information, mais sans redondance.

b) Les problèmes soulevés par une mauvaise modélisation

Attention

Il y a toujours plusieurs façons de modéliser conceptuellement un problème, certaines sont bonnes et d'autres mauvaises. C'est l'expertise de l'ingénieur en charge de la modélisation, à travers son expérience accumulée et sa capacité à traduire le problème posé, qui permet d'obtenir de bons modèles conceptuels.

S'il est difficile de définir un bon modèle conceptuel, on peut en revanche poser qu'un bon modèle logique relationnel est un modèle où la redondance est contrôlée.

On peut alors poser qu'un bon modèle conceptuel est un modèle conceptuel qui conduit à un bon modèle relationnel, après application des règles de passage E-A ou UML vers relationnel. Mais on ne sait pas pour autant le critiquer avant ce passage, autrement qu'à travers l'œil d'un expert.

A défaut de disposer d'outils systématiques pour obtenir de bons modèles conceptuels, on cherche donc à critiquer les modèles relationnels obtenus.

La théorie de la normalisation est une théorie qui permet de critiquer, puis d'optimiser, des modèles relationnels, de façon à en contrôler la redondance.

Exemple : Un mauvais modèle relationnel

Imaginons que nous souhaitions représenter des personnes, identifiées par leur numéro de sécurité sociale, caractérisées par leur nom, leur prénom, ainsi que les véhicule qu'elles ont acheté, pour un certain prix et à une certaine date, sachant qu'un véhicule est caractérisé par son numéro d'immatriculation, un type, une marque et une puissance. On peut aboutir à la représentation relationnelle suivante :

```
1  Personne(NSS, Nom, Prénom, Immat, Marque, Type, Puiss, Date, Prix)
```

Posons que cette relation soit remplie par les données suivantes :

NSS	Nom	Prénom	Immat	Marque	Type	Puiss	Date	Prix
16607...	Dupont	Paul	AJ600AQ	Renault	Clio	5	1/1/96	60000
16607...	Dupont	Paul	AA751KK	Peugeot	504	7	2/7/75	47300
24908...	Martin	Marie	AA751KK	Peugeot	504	7	1/10/89	54900
15405...	Durand	Olivier	AA751KK	Peugeot	504	7	8/8/90	12000
15405...	Durand	Olivier	AJ600AQ	Renault	Clio	5	7/6/98	65000
15405...	Durand	Olivier	XX100XX	BMW	520	10	4/5/01	98000

Tableau 52 Relation redondante

On peut alors se rendre compte que des redondances sont présentes, si l'on connaît NSS on connaît Nom et Prénom, si on connaît Immat, on connaît Marque, Type et Puiss.

NSS	Nom	Prénom	Immat	Marque	Type	Puiss	Date	Prix
16607...	Dupont	Paul	AJ600AQ	Renault	Clio	5	1/1/96	60000
16607...	Dupont	Paul	AA751KK	Peugeot	504	7	2/7/75	47300
24908...	Martin	Marie	AA751KK	Peugeot	504	7	1/10/89	54900
15405...	Durand	Olivier	AA751KK	Peugeot	504	7	8/8/90	12000
15405...	Durand	Olivier	AJ600AQ	Renault	Clio	5	7/6/98	65000
15405...	Durand	Olivier	XX100XX	BMW	520	10	4/5/01	98000

Relation redondante

On sait que ces redondances conduiront à des problèmes de contrôle de la cohérence de l'information (erreur dans la saisie d'un numéro de sécurité sociale), de mise à jour (changement de nom à reporter dans de multiples tuples), de perte d'information lors de la suppression de données (disparition des informations concernant un type de véhicule) et de difficulté à représenter certaines informations (un type de véhicule sans propriétaire).

Complément

On conseillera de lire le chapitre 2 de SQL2 SQL3, applications à Oracle [Delmal01] (pages 42 à 49) qui propose une très bonne démonstration par l'exemple des problèmes posés par une mauvaise modélisation relationnelle.

c) Principes de la normalisation

Fondamental

La théorie de la normalisation est une théorie destinée à concevoir un bon schéma d'une base de données sans redondance d'information et sans risques d'anomalie de mise à jour. Elle a été introduite dès l'origine dans le modèle relationnel.

La théorie de la normalisation est fondée sur deux concepts principaux :

- **Les dépendances fonctionnelles**
  Elles traduisent des contraintes sur les données.
- **Les formes normales**
  Elles définissent des relations bien conçues.

La mise en œuvre de la normalisation est fondée sur la décomposition progressive des relations jusqu'à obtenir des relations normalisées.

Méthode

Afin de mener une bonne conception on cherchera à obtenir un modèle relationnel en BCNF pour lequel l'absence de redondance est simple à démontrer (car les DF sont simples à vérifier).

2. Dépendances fonctionnelles

Objectifs

Savoir repérer et exprimer des dépendances fonctionnelles.
Définir une clé par les dépendances fonctionnelles.

a) Exercice : A1, dans l'eau !

Considérons le schéma de la relation suivante :

- `r (A, B, C, D, E)`

Cette relation est définie en extension par les tuples suivants :

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

Parmi les dépendances fonctionnelles suivantes, lesquelles s'appliquent à r ?

☐	E→D
☐	D→E
☐	C→A
☐	E→B
☐	E→A
☐	B→C
☐	B→D
☐	B→A

b) Dépendance fonctionnelle

Définition : Dépendance fonctionnelle

Soient R(A1, A2, ... , An) un schéma de relation, X et Y des sous-ensembles de A1, A2, ... , An. On dit que X détermine Y, ou que Y dépend fonctionnellement de X, si et seulement s'il existe une fonction qui à partir de toute valeur de X détermine une valeur unique de Y.

Plus formellement on pose que X détermine Y pour une relation R si et seulement si quelle que soit l'instance r de R, alors pour tous tuples t1 et t2 de r on a :

Projection (t1,X) = Projection (t2,X) ⇒ Projection (t1,Y) = Projection (t2,Y)

Syntaxe

Si X détermine Y, on note : X→Y

Exemple

Soit la relation R suivante :

```
1 Personne(NSS, Nom, Prénom, Marque, Type, Puiss, Date, Prix)
```

On peut poser les exemples de DF suivants :

- NSS→Nom
- NSS→Prénom
- Type→Marque
- Type→Puiss
- (NSS, Type, Date)→Prix
- etc.

Remarque : Comment trouver les DF ?

Une DF est définie sur l'intension du schéma et non son extension. Une DF traduit une certaine perception de la réalité. Ainsi la DF (NSS, Type, Date)→Prix signifie que personne n'achète deux voitures du même type à la même date.

La seule manière de déterminer une DF est donc de regarder soigneusement ce que signifient les attributs et de trouver les contraintes qui les lient dans le monde réel.

Remarque : Pourquoi trouver les DF ?

Les DF font partie du schéma d'une BD, en conséquence, elles doivent être

déclarées par les administrateurs de la BD et être contrôlées par le SGBD.

De plus l'identification des DF est la base indispensable pour déterminer dans quelle forme normale est une relation et comment en diminuer la redondance.

c) Axiomes d'Armstrong

Introduction

Les DF obéissent à des propriétés mathématiques particulières, dites axiomes d'Armstrong.

Définition : Réflexivité

Tout groupe d'attributs se détermine lui même et détermine chacun de ses attributs (ou sous groupe de ses attributs).

Soient X et Y des attributs :

XY→XY et XY→X et XY→Y

Définition : Augmentation

Si un attribut X détermine un attribut Y, alors tout groupe composé de X enrichi avec d'autres attributs détermine un groupe composé de Y et enrichi des mêmes autres attributs.

Soient X, Y et Z des attributs :

X→Y ⇒ XZ→YZ

Définition : Transitivité

Si un attribut X détermine un attribut Y et que cet attribut Y détermine un autre attribut Z, alors X détermine Z.

Soient X, Y et Z des attributs :

X→Y et Y→Z ⇒ X→Z

d) Autres propriétés déduites des axiomes d'Armstrong

Introduction

A partir des axiomes d'Amstrong, on peut déduire un certain nombre de propriétés supplémentaires.

Définition : Pseudo-transitivité

Si un attribut X détermine un autre attribut Y, et que Y appartient à un groupe G qui détermine un troisième attribut Z, alors le groupe G' obtenu en substituant Y par X dans G détermine également Z.

Soient, W, X, Y et Z des attributs :

X→Y et WY→Z ⇒ WX→Z

Cette propriété est déduite de l'augmentation et de la transitivité :

X→Y et WY→Z ⇒ WX→WY et WY→Z ⇒ WX→Z

Définition : Union

Si un attribut détermine plusieurs autres attributs, alors il détermine tout groupe composé de ces attributs.

Soient X, Y et Z des attributs :

X→Y et X→Z ⇒ X→YZ

Cette propriété est déduite de la réflexivité, de l'augmentation et de la transitivité :

X→Y et X→Z ⇒ X→XX et XX→XY et YX→YZ ⇒ X→YZ

Définition : Décomposition

Si un attribut détermine un groupe d'attribut, alors il détermine chacun des attributs de ce groupe pris individuellement.

Soient X, Y et Z des attributs :

X→YZ ⇒ X→Z et X→Y

Cette propriété est déduite de la réflexivité et de la transitivité :

X→YZ ⇒ X→YZ et YZ→Z ⇒ X→Z

e) DF élémentaire

Définition : Dépendance fonctionnelle élémentaire

Soit G un groupe d'attributs et A un attribut, une DF G→A est élémentaire si A n'est pas incluse dans G et s'il n'existe pas d'attribut A' de G qui détermine A.

Exemple : DF élémentaires

- AB→C est élémentaire si ni A, ni B pris individuellement ne déterminent C.
- Nom, DateNaissance, LieuNaissance→Prénom est élémentaire.

Exemple : DF non élémentaires

- AB→A n'est pas élémentaire car A est incluse dans AB.
- AB→CB n'est pas élémentaire car CB n'est pas un attribut, mais un groupe d'attributs.
- N°SS→Nom, Prénom n'est pas élémentaire.

Remarque

On peut toujours réécrire un ensemble de DF en un ensemble de DFE, en supprimant les DF triviales obtenues par réflexivité et en décomposant les DF à partie droite non atomique en plusieurs DFE.

Exemple : Réécriture de DF en DFE

On peut réécrire les DF non élémentaires de l'exemple précédent en les décomposant DFE :

- AB→A n'est pas considérée car c'est une DF triviale obtenu par réflexivité.
- AB→CB est décomposée en AB→C et AB→B, et AB→B n'est plus considérée car triviale.
- N°SS→Nom, Prénom est décomposée en N°SS→Nom et N°SS→Prénom.

f) Fermeture transitive des DFE

Définition : Fermeture transitive

On appelle fermeture transitive F+ d'un ensemble F de DFE, l'ensemble de toutes les DFE qui peuvent être composées par transitivité à partir des DFE de F.

Exemple

Soit l'ensemble F = {A→B, B→C, B→D, A→E}.
La fermeture transitive de F est F+ = { A→B, B→C, B→D, A→E, A→C, A→D }

g) Couverture minimale des DFE

Définition : Couverture minimale

La couverture minimale d'un ensemble de DFE est un sous-ensemble minimum des DFE permettant de générer toutes les autres DFE.
Synonymes : Famille génératrice

Remarque

Tout ensemble de DFE (et donc tout ensemble de DF) admet au moins une couverture minimale (et en pratique souvent plusieurs).

Exemple

L'ensemble F = {A→B, A→C, B→C, C→B} admet les deux couvertures minimales :
CM1 = {A→C, B→C, C→B} et CM2 = {A→B, B→C, C→B}

h) Graphe des DFE

On peut représenter un ensemble de DFE par un graphe orienté (ou plus précisément un réseau de Pétri), tel que les nœuds sont les attributs et les arcs les DFE (avec un seul attribut en destination de chaque arc et éventuellement plusieurs en source).

Exemple : Relation Voiture

Soit la relation Voiture(NVH, Marque, Type, Puis, Couleur) avec l'ensemble des DF F = {NVH→Type, Type→Marque, Type→Puis, NVH→Couleur}. On peut représenter F par le graphe ci-dessous :

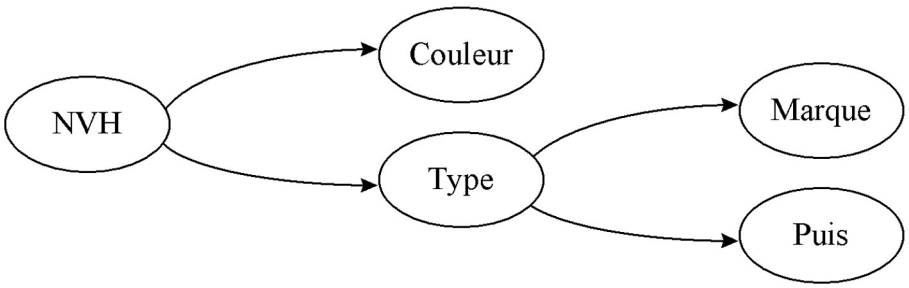


Image 21 Graphe des DFE de la relation Voiture

Exemple : Relation CodePostal

Soit la relation CodePostal(Code, Ville, Rue) avec l'ensemble des DF F={Code→Ville, (Ville,Rue)→Code}. On peut réprésenter F par le graphe ci-dessous :

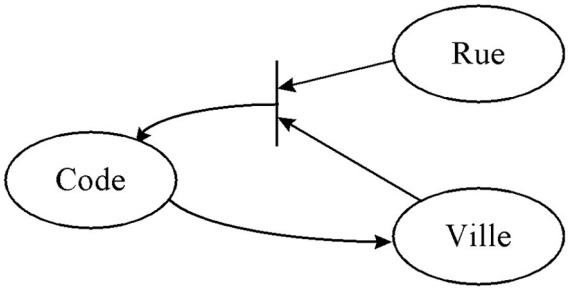


Image 22 Graphe des DFE de la relation CodePostal

i) Définition formelle d'une clé

Définition : Clé

Soient une relation R(A1,A2,...,An) et K un sous-ensemble de A1,A2,... ,An.

K est une clé de R si et seulement si :

1. K→A1,A2,...,An
2. et il n'existe pas X inclus dans K tel que X→A1,A2,...,An.

Fondamental

Une clé est donc un ensemble minimum d'attributs d'une relation qui détermine tous les autres.

Remarque : Clés candidates et clé primaire

Si une relation comporte plusieurs clés, chacune est dite clé candidate et l'on en choisit une en particulier pour être la clé primaire.

Attention : Les clés candidates sont des clés !

Toutes les clés candidates sont des clés, pas seulement la clé primaire.

Remarque : Les clés candidates se déterminent mutuellement

Toute clé candidate détermine les autres clés candidates, puisque qu'une clé détermine tous les attributs de la relation.

Complément : Relation "toute clé"

Étant donné qu'une relation dispose forcément d'une clé, si une relation R n'admet aucune clé K sous ensemble des attributs A1..An de R, alors c'est que K=A1..An (la clé est composée de **tous** les attributs de R).

On parle de relation "toute clé".

j) Exercice : A1, touché !

Considérons le schéma de la relation suivante :

- `r (A, B, C, D, E)`

Cette relation est définie en extension par les tuples suivants :

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

Après avoir énoncé les DF, déterminer, parmi les groupes d'attributs suivants, lesquels sont des clés ?

- ☐ A
- ☐ B
- ☐ C
- ☐ D
- ☐ E
- ☐ {B,E}
- ☐ {A,B,C,D,E}

3. Formes normales

Objectifs

Connaître les formes normales et leurs implications en terme de redondance.

a) Les formes normales

Les formes normales ont pour objectif de définir la décomposition des schémas relationnels, tout en préservant les DF et sans perdre d'informations, afin de représenter les objets et associations canoniques du monde réel de façon non redondante.

On peut recenser les 6 formes normales suivantes, de moins en moins redondantes :

- la première forme normale
- la deuxième forme normale
- la troisième forme normale
- la forme normale de Boyce-Codd

- la quatrième forme normale
- la cinquième forme normale

La troisième forme normale est généralement reconnue comme la plus importante à respecter.

La BCNF est la plus simple à établir.

b) Principe de la décomposition

L'objectif de la décomposition est de "casser" une relation en relations plus petites afin d'en éliminer les redondances et sans perdre d'information.

Définition : Décomposition

La décomposition d'un schéma de relation R(A1,A2,...,An) est le processus de remplacement de ce schéma par une collection de schémas R1,R2,...,Rn telle qu'il est possible de reconstruire R par des opérations relationnelles de jointure sur R1,R2,...,Rn.

Définition : Décomposition préservant les DF

Une décomposition d'une relation R en relations R1,R2,...Rn préserve les DF si la fermeture transitive F+ des DF de R est la même que la fermeture transitive F+ de l'union des DF de R1,R2,...,Rn.

Exemple : Décomposition préservant les DF d'une relation Voiture

Soit la relation Voiture(Numéro,Marque,Type,Puissance,Couleur) avec la fermeture transitive suivante :

- Numéro→Marque
- Numéro→Type
- Numéro→Puissance
- Numéro→Couleur
- Type→Marque
- Type→Puissance

On peut décomposer Voiture en préservant les DF en deux relations R1(Numéro,Type,Couleur) et R2(Type,Puissance,Marque).

c) Première forme normale

Définition : 1NF

Une relation est en 1NF si elle possède au moins une clé et si tous ses attributs sont atomiques.

Définition : Attribut atomique

Un attribut est atomique si il ne contient qu'une seule valeur pour un tuple donné, et donc s'il ne regroupe pas un ensemble de plusieurs valeurs.

Exemple : Avoir plusieurs métiers

Soit la relation Personne instanciée par deux tuples :

```
1 Personne(#Nom, Profession)
```

```
1 (Dupont, Géomètre)
2 (Durand, Ingénieur-Professeur)
```

La relation n'est pas en 1NF, car l'attribut Profession peut contenir plusieurs valeurs.

Pour que la relation soit en 1NF, on pourrait par exemple ajouter Profession à la clé et faire apparaître deux tuples pour Durand, on obtiendrait :

```
1 Personne(#Nom, #Profession)
```

```
1 (Dupont, Géomètre)
2 (Durand, Ingénieur)
3 (Durand, Professeur)
```

Une autre solution aurait été d'ajouter un attribut ProfessionSecondaire. On obtiendrait ainsi :

```
1 Personne(#Nom, Profession, ProfessionSecondaire)
```

```
1 (Dupont, Géomètre, Null)
2 (Durand, Ingénieur, Professeur)
```

Remarque : Relativité de la notion d'atomicité

L'atomicité d'un attribut est souvent relative : on peut décider qu'un attribut contenant une date n'est pas atomique (et que le jour, le mois et l'année constituent chacun une valeur), ou bien que l'attribut est de domaine date et donc qu'il est atomique.

Fondamental : Énoncer les clés

Le modèle relationnel impose qu'une relation ait une clé, donc la condition "est en 1NF si elle possède une clé" est superflue (au pire la relation est toute clé).

Il est néanmoins fondamental d'avoir identifié **toutes** les clés au début du processus de normalisation.

d) Deuxième forme normale

Définition : 2NF

Une relation est en 2NF si elle est en 1NF et si tout attribut n'appartenant à aucune clé candidate ne dépend pas d'une partie seulement d'une clé candidate.

Exemple : Echelle de salaire

Soit la relation Personne :

```
1 Personne(#NumeroEmployé, #Profession, Nom, Prénom, Salaire)
```

Soit les DF suivantes sur cette relation :

- NumeroEmployé, Profession→Nom
- NumeroEmployé, Profession→Prénom
- NumeroEmployé, Profession→Salaire
- Profession→Salaire

Personne n'est pas en 2NF car Profession (une partie de clé) détermine Salaire (un attribut qui n'appartient pas à une clé)

Pour avoir un schéma relationnel en 2NF, il faut alors décomposer Personne en deux relations :

```
1 Personne(#NumeroEmployé, #Profession=>Profession, Nom, Prenom)
2 Profession(#Profession, Salaire)
```

On remarque que ce schéma est en 2NF (puisque Salaire dépend maintenant fonctionnellement d'une clé et non plus d'une partie de clé).

On remarque aussi que la décomposition a préservé les DF, puisque nous avons à présent :

- Profession→Salaire (DF de la relation Profession)
- NumeroEmployé, Profession→Profession (par Réflexivité)
- NumeroEmployé, Profession→Salaire (par Transitivité)

Attention

La définition de la 2NF doit être vérifiée pour toutes les clés candidates et non seulement la clé primaire (dans le cas où il y a plusieurs clés).

Remarque

Si toutes les clés d'une relation ne contiennent qu'un unique attribut, et que la relation est en 1NF, alors la relation est en 2NF.

e) Troisième forme normale

Définition : 3NF

Une relation est en 3NF si elle est en 2NF et si tout attribut n'appartenant à aucune clé candidate ne dépend directement que de clés candidates.

C'est à dire que toutes les DFE vers des attributs n'appartenant pas à une clé, sont issues d'une clé.

Exemple : Échelle de salaire et de prime

Soit la relation Profession :

```
1 Profession(#Profession, Salaire, Prime)
```

Soit les DF suivantes sur cette relation :

- Profession→Salaire
- Profession→Prime
- Salaire→Prime

Cette relation n'est pas en 3NF car Salaire, qui n'est pas une clé, détermine Prime.

Pour avoir un schéma relationnel en 3NF, il faut décomposer Profession :

```
1 Profession(#Profession, Salaire=>Salaire)
2 Salaire(#Salaire, Prime)
```

Ce schéma est en 3NF, car Prime est maintenant déterminé par une clé.

On remarque que cette décomposition préserve les DF, car par transitivité, Profession détermine Salaire qui détermine Prime, et donc Profession détermine toujours Prime.

Attention : Clé candidate

La définition concerne toutes les clés candidates et non uniquement la clé primaire (SQL avancé : Programmation et techniques avancées [Celko00], p.27).

Fondamental

Il est souhaitable que les relations logiques soient en 3NF. En effet, il existe toujours une décomposition sans perte d'information et préservant les DF d'un schéma en 3NF. Si les formes normales suivantes (BCNF, 4NF et 5NF) assurent un niveau de redondance encore plus faible, la décomposition permettant de les atteindre ne préserve plus systématiquement les DF.

Remarque : Limite de la 3NF

Une relation en 3NF permet des dépendances entre des attributs n'appartenant pas à une clé vers des parties de clé.

Complément : 3NF et 2NF

Une relation en 3NF est forcément en 2NF car :

- Toutes les DFE vers des attributs n'appartenant pas à une clé sont issues d'une clé, ce qui implique qu'il n'existe pas de DFE, issues d'une partie de clé vers un attribut qui n'appartient pas à une clé.
- Il ne peut pas non plus exister de DFE issues d'une partie de clé vers un attribut appartenant à une clé, par définition de ce qu'une clé est un ensemble minimum.

On n'en conclut qu'il ne peut exister de DFE, donc a fortiori pas de DF, issues d'une partie d'une clé, et donc que toutes les DF issues d'une clé sont élémentaires.

f) Forme normale de Boyce-Codd

Définition : BCNF

Une relation est en BCNF si elle est en 3NF et si les seules DFE existantes sont celles pour lesquelles une clé candidate détermine un attribut.

Exemple : Employés

Soit la relation Personne :

```
1 Personne(#N°SS, #Pays, Nom, Région)
```

Soit les DF suivantes sur cette relation :

- N°SS,Pays→Nom
- N°SS,Pays→Région
- Région→Pays

Il existe une DFE qui n'est pas issue d'une clé et qui détermine un attribut appartenant à une clé. Cette relation est en 3NF, mais pas en BCNF (car en BCNF toutes les DFE sont issues d'une clé).

Pour avoir un schéma relationnel en BCNF, il faut décomposer Personne :

```
1 Personne(#N°SS, #Region=>Region, Nom)
2 Region(#Region, Pays)
```

Remarquons que les DF n'ont pas été préservées par la décomposition puisque

N°SS et Pays ne déterminent plus Région.

Fondamental : Simplicité

La BCNF est la forme normale la plus facile à appréhender intuitivement et formellement, puisque les seules DFE existantes sont de la forme K→A où K est une clé.

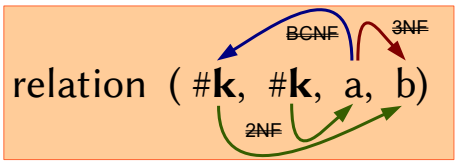
Méthode : À retenir

Pour prouver qu'une BD n'est pas redondante, on pourra se contenter de vérifier qu'elle est en BCNF.

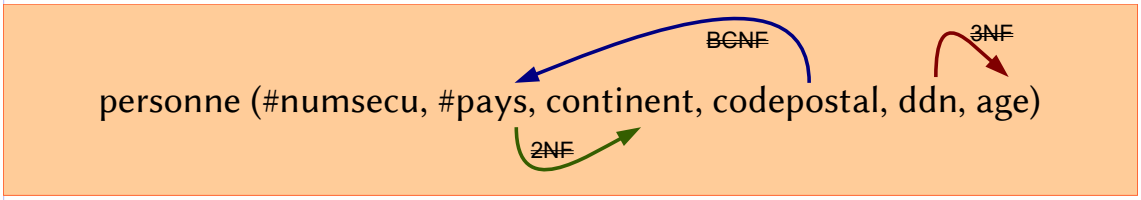
Attention : Non préservation des DF

Une décomposition en BCNF ne préserve pas toujours les DF.

g) Synthèse



Exemple



h) Exercice : A1, coulé !

Considérons le schéma de la relation suivante :

- `r (A, B, C, D, E)`

Cette relation est définie en extension par les tuples suivants :

A	B	C	D	E
a1	b2	c2	d3	e2
a1	b2	c2	d1	e4
a2	b3	c2	d1	e5
a2	b4	c5	d1	e5

Après avoir énoncé les DF et les clés, déterminer la forme normale du schéma ?

- ☐ 1NF
- ☐ 2NF
- ☐ 3NF
- ☐ BCNF

4. Bibliographie commentée sur la normalisation

Complément : Synthèses

SQL2 SQL3, applications à Oracle [Delmal01]

On conseillera de lire le chapitre 2 (pages 42 à 49) qui propose une très bonne démonstration par l'exemple des problèmes posés par une mauvaise modélisation relationnelle.

Une description claire des formes normales, rendue simple et pratique grâce à des exemples représentatifs (chapitre 2).

B. Exercices

1. De quoi dépend un cours ?

[15 min]

On considère le schéma relationnel R défini sur les attributs suivants : C : cours, P : professeur, H : heure, S : salle, E : étudiant, N : note.

Un nuplet (c, p, h, s, e, n) a pour signification que le cours c est fait par le professeur p à l'heure h dans la salle s pour l'étudiant e qui a reçu la note n.

L'ensemble E des dépendances fonctionnelles initiales est le suivant :

- C → P
- H,S → C

- H,P → S
- C,E → N
- H,E → S

Question 1

Donner la fermeture transitive F+ des dépendances fonctionnelles élémentaires engendrées par E.

Question 2

Quelle est la clé de la relation R ? Montrer qu'elle est unique.

2. Cuisines et dépendances

[20 min]

On considère une relation R construite sur les attributs Propriétaire, Occupant, Adresse, Noapt, Nbpièces, Nbpersonnes, un nuplet (p, o, a, n, nb1, nb2) ayant la signification suivante : La personne o habite avec nb2 personnes l'appartement de numéro n ayant nb1 pièces dont le propriétaire est p.

Une analyse de cette relation nous fournit un ensemble initial E de dépendances fonctionnelles :

- occupant → adresse
- occupant → noapt
- occupant → nbpersonnes
- adresse, noapt → propriétaire
- adresse, noapt → occupant
- adresse, noapt → nbpièces

Question 1

Donner l'ensemble des DFE engendrées par E (fermeture transitive F+).

Question 2

Quelles sont les clés candidates de R ?

Question 3

Montrer que R est en 3NF de deux façons différentes (sans passer et en passant par la BCNF).

3. Test : Normalisation

Exercice 1

Soit la relation suivante et une couverture minimale des DF associée.

```
1 tUtilisateur (pklogin,mdp,nom,prenom,ville)
```

- *pklogin→mdp,nom,prenom,ville*
- *nom,prenom→pklogin*
- *ville→nom*

Sélectionner la ou les clés de cette relation.

☐	pklogin
☐	mdp
☐	nom
☐	prenom
☐	ville
☐	(pklogin, mdp)
☐	(pklogin, nom)
☐	(nom, prenom)
☐	(ville, nom)
☐	(nom, prenom, ville)

Exercice 2

Soit le schéma relationnel (on pose que les attributs A, B, C, D, X et Y sont atomiques) :

```
1 R1(A, B, C, D)
2 R2(X, Y)
```

Soit les dépendances fonctionnelles identifiées :

- *A → C*
- *A,B → D*
- *X → Y*

En quelles formes normales est ce schéma relationnel ?

☐	1NF
☐	2NF
☐	3NF
☐	BCNF

Exercice 3

Soit le schéma relationnel suivant :

```
1 Personne (Nom, Prenom, Age, DateNaissance)
```

Soit les DF suivantes :

- *Nom → Prenom*
- *Nom, Prenom → Age*
- *Prenom → DateNaissance, Nom*
- *DateNaissance, Age → Age, Nom*
- *Age, Nom → Age, DateNaissance*
- *DateNaissance → Age*

Quelles sont les clés candidates pour la relation Personne ?

- [ ] Nom
- [ ] Prenom
- [ ] Age
- [ ] DateNaissance

Exercice 4

Soit la relation R (A:Int, B:Int, C:Int, D:Int, E:Int) et l'ensemble de DFE *suivant : {A → B ; A → C ; A → D ; A → E ; B → A ; B → C}*

Sélectionner toutes les assertions vraies.

- [ ] A est une clé
- [ ] B est une clé
- [ ] C est une clé
- [ ] Le schéma est en 1NF
- [ ] Le schéma est en 2 NF
- [ ] Le schéma est en 3 NF
- [ ] Cet ensemble de DFE est une fermeture transitive.
- [ ] Cet ensemble de DFE est une couverture minimale.

Exercice 5

Soit le schéma relationnel (on pose que tous les attributs sont atomiques) :

```
1 Adresse (Numero, Rue, Ville=>Ville, Pays=>Ville)
2 Ville (Ville, Pays=>Pays)
3 Pays (Pays)
```

En quelles formes normales est ce schéma relationnel ?

- [ ] 1NF
- [ ] 2NF
- [ ] 3NF
- [ ] BCNF

Conception de bases de données normalisées

VIII

A. Cours

1. Conception de bases de données normalisées

Objectifs

Savoir créer des schémas relationnels en troisième forme normale.

a) Exercice : Étapes de la conception d'une base de données

Mettre dans l'ordre les étapes de conception suivantes.

1. Décomposition des relations jusqu'à arriver en 3NF en préservant les DF
2. Élaboration du modèle logique en relationnel ou relationnel-objet
3. Énonciation de la forme normale avant normalisation
4. Établissement des DF sous la forme de la fermeture transitive pour chaque relation du modèle
5. Modélisation conceptuelle en UML ou E-A
6. Création du code SQL LDD pour un SGBDR ou un SGBDRO
7. Analyse de la situation existante et des besoins
8. Rétro-conception d'un modèle UML normalisé

Réponse : ___ ___ ___ ___ ___ ___ ___ ___

b) Algorithme de décomposition 0NF->1NF

Méthode : Attributs composés

Soit R une relation. Si R contient un attribut non atomique portant sur des valeurs

hétérogènes (correspondant à un attribut composé au niveau conceptuel), alors l'attribut est décomposé en plusieurs attributs $a_1...a_n$.

`R(#pk,a,b,...)` avec `a` non atomique se décompose en : `R(#pk,a1,a2,...,an,b,...)`

Méthode : Attributs multivalués

Soit R une relation avec la clé primaire pk. Si R contient un attribut non atomique portant sur des valeurs homogènes (correspondant à un attribut multivalué au niveau conceptuel), alors R est décomposée en R1 et R2, tel que :

- R1 est R moins l'attribut a
- R2 est composé de pk et de a, avec (pk,a) clé primaire de R2 et R2.pk clé étrangère vers R1.pk

`R(#pk,a,b,...)` avec `a` non atomique se décompose en :

- `R1(#pk,b,...)`
- `R2(#pk=>R1,#a)`

c) Exemple de décomposition 0NF->1NF (attribut multivalué)

Exemple : Situation initiale

```
1  Élève(#login, nom, prénom, uvs)
```

Exemple d'enregistrement :

```
1  ('tpassoir', 'Passoire', 'Toto', ('SY02','NF26','NF17'))
```

Méthode

L'attribut `uvs` est multivalué (donc non atomique). On va donc créer deux relations, la première sans l'attribut `uvs`, et la seconde qui va contenir autant d'enregistrements que de valeur par enregistrement dans l'enregistrement initial.

Exemple : Situation finale

```
1  Élève(#login, nom, prénom)
2  UVÉlève(#login=>Élève, #uv)
```

En reprenant le même exemple, on va avoir l'enregistrement suivant dans `Élève` :

```
1  ('tpassoir', 'Passoire', 'Toto')
```

et les enregistrements suivants dans `UVÉlève` :

```
1  ('tpassoir', 'SY02')
2  ('tpassoir', 'NF26')
3  ('tpassoir', 'NF17')
```

d) Algorithme de décomposition 1NF->3NF

Fondamental : Décomposition > 1NF

Pour les NF supérieures à 1, afin de normaliser une relation R on réalise une décomposition en R1 et R2 pour chaque DFE responsable d'un défaut de normalisation tel que :

- la partie gauche de la DFE :
  a. devient la clé primaire de R2
  b. devient une clé étrangère de R1 vers R2
- la partie droite de la DFE
  a. est enlevée de R1
  b. est ajoutée comme attributs simples de R2

Méthode : Décomposition 1NF->2NF

Soit R une relation comportant une clé composée de k1 et k1'. Si R contient une DFE de k1' vers des attributs n'appartenant pas à la clé, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins les attributs déterminés par k1' et avec k1' clé étrangère vers R2
- R2 est composée de k1' et des attributs qu'elle détermine, avec k1' clé primaire de R2

`R(#pk,k1,k1',a,b,c,...)` avec (k1,K1') clé et k1'→a,b se décompose en :

- `R1(#pk,k1,k1'=>R2,c,...)`
- `R2(#k1',a,b)`

Méthode : Décomposition 2NF->3NF

Soit R une relation comportant une DFE de a vers b qui n'appartiennent pas à une clé, alors R est décomposée en R1 et R2, tel que :

- R1 est R moins les attributs déterminés par a et avec a clé étrangère vers R2
- R2 est composée de a et des attributs qu'elle détermine, avec a clé primaire de R2

`R(#pk,a,b,c,...)` avec a→b se décompose en

- `R1(#pk,a=>R2,c,...)`
- `R2(#a,b)`

e) Exemple de décomposition 1NF->3NF

Exemple : Situation initiale

```
1  Resultat (#pknum,knumetu,kuv,prenom,nom,credits,resultat,obtenu) avec
   (knumetu,kuv) clé
```

DF :

- pknum → knumetu,kuv,prenom,nom,credits,resultat,obtenu
- knumetu,kuv → pknum,prenom,nom,credits,resultat,obtenu
- **knumetu → prenom,nom**
- **kuv → credits**
- **resultat → obtenu**

La relation est en 1NF.

pknum	knumetu	kuv	prenom	nom	credits	resultat	obtenu
1	X01	NF17	Pierre	Alpha	6	A	oui
2	X01	NF26	Pierre	Alpha	6	B	oui
3	X02	NF17	Alphonse	Béta	6	F	non

Tableau 53 Exemple d'instance

knumetu → prenom,nom

```
1 Resultat (#pknum,knumetu=>Etudiant,kuv,credits,resultat,obtenu)
2 Etudiant (#knumetu,prenom,nom)
```

kuv → credits

```
1 Resultat (#pknum,knumetu=>Etudiant,kuv=>Uv,resultat,obtenu)
2 Etudiant (#knumetu,prenom,nom)
3 Uv(#kuv,credits)
```

resultat → obtenu

```
1 Resultat (#pknum,knumetu=>Etudiant,kuv=>Uv,resultat=>Note)
2 Etudiant (#knumetu,prenom,nom)
3 Uv(#kuv,credits)
4 Note(#resultat,obtenu)
```

f) Normalisation par transformation d'attributs en méthodes

Il arrive que la fonction sous-jacente à la DF soit une fonction simple, que l'on peut calculer. Par exemple pour la DF : `ddn → age ( ddn)`, on peut calculer *age* en fonction de *ddn* par le calcul : *age = today() - ddn*.

Méthode

Chaque fois que c'est possible, on remplacera un attribut par une méthode si cela permet de supprimer une DF sans perdre d'information.

Cette solution est à privilégier *a priori* sur une décomposition.

En relationnel, on supprimera l'attribut de la table et on ajoutera une vue permettant de le retrouver.

Exemple : Existence d'une fonction de calcul simple

Soit la relation en 2NF :

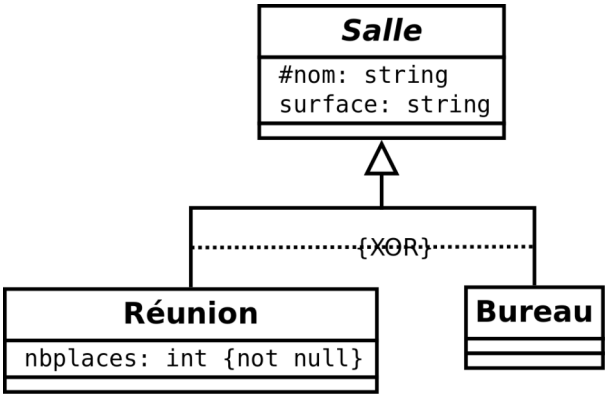
```
1 personne (#numsecu, nom, prenom, ddn, age) avec ddn → age
```

On remplacera cette relation par une relation en 3NF et une vue :

```
1 Personne (#numsecu, nom, prenom, ddn)
2 vPersonne (#numsecu, nom, prenom, ddn, age) avec  age = today() - ddn
```

Exemple : Cas de transformation d'un héritage exclusif par la classe mère

Soit le schéma UML suivant :



Une transformation de l'héritage par la classe mère donnera :

```
1  Salle (#nom:string, surface:string, nbplaces:int, type:{Réunion|
   Bureau})
2  DF : nom → surface, type, nbplaces et nbplaces → type
```

On peut donc supprimer l'attribut type et aboutir à la relation en 3NF :

```
1  Salle (#nom:string, surface:string, nbplaces:int)
2  vSalle (nom, surface, nbplaces, type) avec SI nbplace IS NULL ALORS
   type=Bureau SINON type=Réunion
```

g) Synthèse : Processus de conception d'une base de données normalisée

1. Analyse et clarification du problème posé
2. Modélisation conceptuelle en UML
   Résultat : MCD_1
3. Traduction en relationnel en utilisant les règles de passage UML vers relationnel
   Résultat : MLD_1
4. Établissement des DF sous la forme de la fermeture transitive pour chaque relation du modèle
   Résultat : MLD_1 avec F+
5. Établissement de la forme normale
   Résultat : MLD_1 avec F+ et NF
6. Décomposition des relations jusqu'à arriver en 3NF en préservant les DF
   Résultat : MLD_2 avec F+ et 3NF
7. Rétro-conception du modèle UML correspondant
   Résultat : MCD_2
8. Implémentation en SQL du modèle MLD_2

* *

*

La normalisation permet de décomposer un schéma relationnel afin d'obtenir des

relations non redondantes.

La 3NF est souhaitable car toujours possible à obtenir, sans perte d'information et sans perte de DF. La BCNF est également indiquée, car elle est un peu plus puissante, et plutôt plus simple que la 3NF.

La BCNF n'est pas encore suffisante pour éliminer toutes les redondances. Il existe pour cela les 4NF et 5NF qui ne sont pas abordées dans ce cours. Notons également que les cas de non-4NF et de non-5NF sont assez rares dans la réalité.

2. Exemple de synthèse : MCD-Relationnel-Normalisation-SQL

Problème posé

Soit un modèle conceptuel représentant :

- un type d'entité "chercheur", identifié par le numéro de sécurité sociale, et possédant les autres propriétés suivantes : le nom, le nom de l'université à laquelle il appartient, la ville dans laquelle est basée cette université.
- un type d'entité "professeur", héritant de "chercheur"
- un type d'entité "doctorant", héritant de "chercheur"
- une association de type "encadrement" entre professeur et doctorant (un professeur pouvant encadrer plusieurs doctorants et un doctorant n'ayant qu'un et un seul directeur de thèse).

Afin de réaliser le modèle de données :

1. Dessiner le modèle conceptuel
2. Traduire le modèle conceptuel en modèle logique relationnel.
3. Après avoir identifié les DF, normaliser le modèle relationnel en BCNF.
4. Ecrire les instructions SQL de création d'un tel modèle.

a) Première étape : Modélisation conceptuelle

Méthode : Modélisation conceptuelle : Entité Chercheur

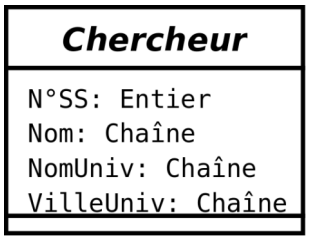


Image 23 Conception UML (1/4)

Méthode : Modélisation conceptuelle : Entité Professeur

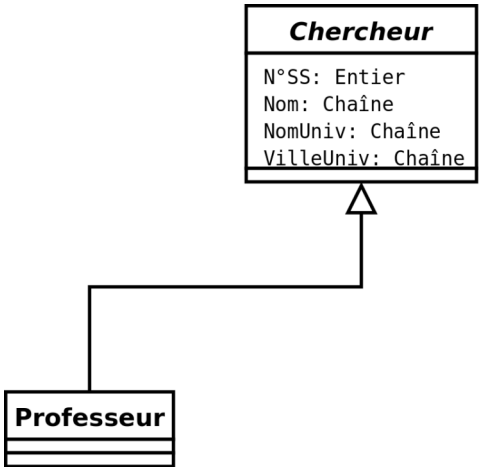


Image 24 Conception UML (2/4)

Méthode : Modélisation conceptuelle : Entité Doctorant

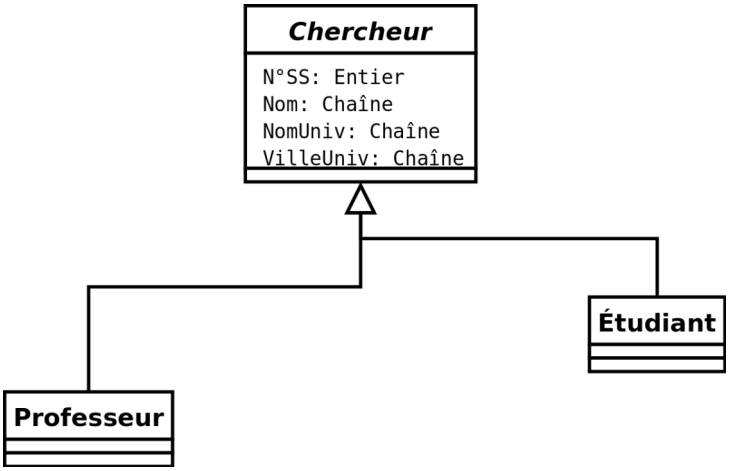


Image 25 Conception UML (3/4)

Méthode : Modélisation conceptuelle : Association Encadrement

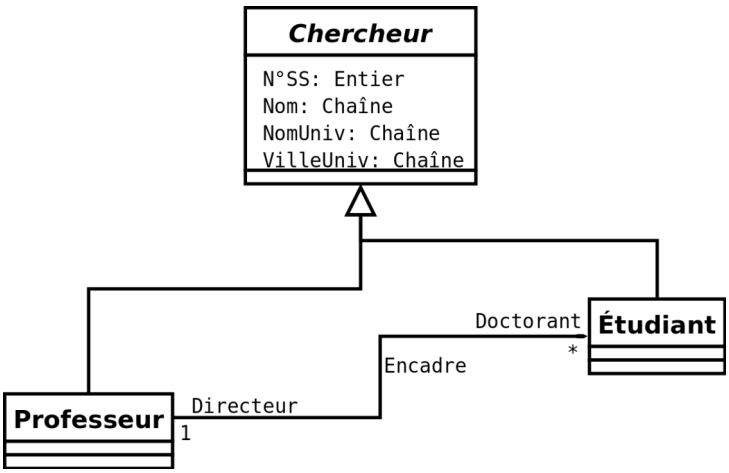


Image 26 Conception UML (4/4)

b) Deuxième étape : Modèle relationnel

Méthode : Modèle relationnel : Héritage

Choix de transformation de l'héritage : L'héritage est exclusif (les professeurs ne sont plus doctorants), mais pas complet, **car l'association Encadre n'est pas symétrique**. On choisit donc un héritage par les classes filles (Chercheur étant par ailleurs abstrait).

Méthode : Modèle relationnel : Entité Professeur

```
1 Professeur (#N°SS:int(13), Nom:char(20), NomUniv:char(50),
  VilleUniv:char(20))
```

Méthode : Modèle relationnel : Entité Doctorant

```
1 Professeur (#N°SS:int(13), Nom:char(20), NomUniv:char(50),
  VilleUniv:char(20))
2 Doctorant (#N°SS:int(13), Nom:char(20), NomUniv:char(50),
  VilleUniv:char(20))
```

Méthode : Modèle relationnel : Association EncadréPar

```
1 Professeur (#N°SS:int(13), Nom:char(20), NomUniv:char(50),
  VilleUniv:char(20))
2 Doctorant (#N°SS:int(13), Nom:char(20), NomUniv:char(50),
  VilleUniv:char(20), EncadrePar=>Professeur)
```

c) Troisième étape : Dépendances fonctionnelles

Méthode : Dépendances fonctionnelles : DF évidente (professeur)

- Professeur.N°SS → Nom, NomUniv, VilleUniv

Méthode : Dépendances fonctionnelles : DF évidente (doctorant)

- Professeur.N°SS → Nom, NomUniv, VilleUniv
- Doctorant.N°SS → Nom, NomUniv, VilleUniv, EncadrePar

Méthode : Dépendances fonctionnelles : DF déduites du sens des propriétés

Connaissant l'université, on connaît la ville, donc :

- Professeur.N°SS → Nom, NomUniv, VilleUniv
- Professeur.NomUniv → VilleUniv
- Doctorant.N°SS → Nom, NomUniv, VilleUniv, EncadrePar
- Doctorant.NomUniv → VilleUniv

d) Quatrième étape : Formes normales

Méthode : Forme normale : 1NF

Le schéma est en 1NF (clés et attributs atomique).

Méthode : Forme normale : 2NF

Le schéma est en 2NF (la clé est composée d'un seul attribut)

Méthode : Forme normale : 3NF

La schéma n'est pas en 3NF : NomUniv → VilleUniv

e) Cinquième étape : Décomposition

Méthode : Normalisation en BCNF : Résultat

```
1  Professeur (#N°SS:int(13), Nom:char(20), NomUniv=>Univ)
2  Doctorant (#N°SS:int(13), Nom:char(20), NomUniv=>Univ,
    EncadrePar=>Professeur)
3  Univ (#NomUniv:char(50), VilleUniv:char(20))
```

Méthode : Normalisation en BCNF : Vérification

Le modèle est bien en BCNF, toutes les DF ont pour source une clé.

Méthode : Normalisation en BCNF : Conservation des DF

La transformation préserve les DF car :

- N°SS → NomUniv et Univ.Nom → Ville
- Donc N°SS → Univ.Ville (par transitivité)

f) Sixième étape : Implémentation SQL LDD

Méthode : Implémentation SQL : Professeur

```
1  CREATE TABLE Professeur (
2  Numss VARCHAR(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL);
```

Méthode : Implémentation SQL : Doctorant

```
1  CREATE TABLE Professeur (
2  Numss VARCHAR(13) PRIMARY KEY,
3  Nom CHAR(20) NOT NULL);
4
5  CREATE TABLE  Doctorant (
6  Numss VARCHAR(13) PRIMARY KEY,
7  Nom CHAR(20) NOT NULL,
8  EncadrePar VARCHAR(13) REFERENCES Professeur(Numss));
```

Méthode : Implémentation SQL : Univ

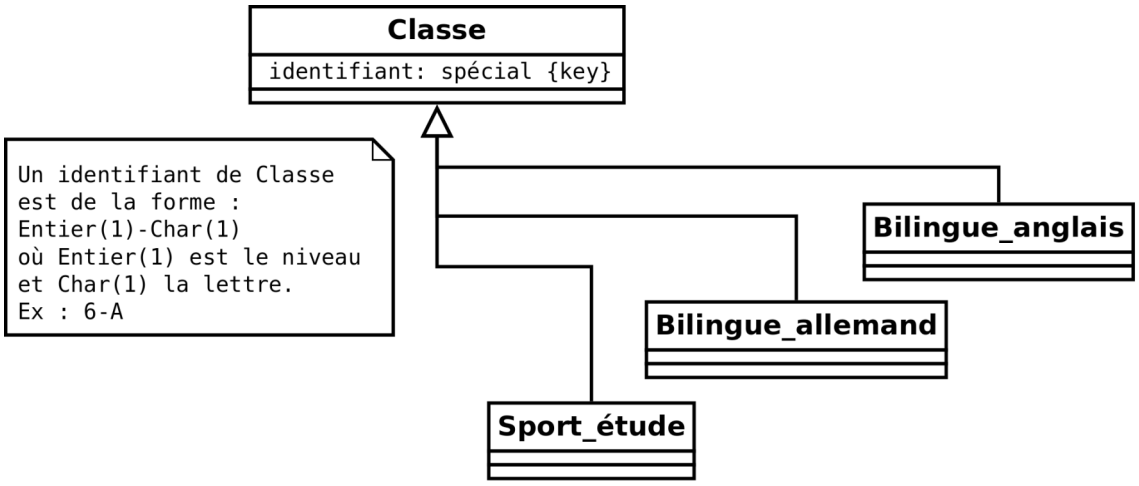
```
1  CREATE TABLE Univ (
2  Nom CHAR(50) PRIMARY KEY,
3  Ville CHAR(20) );
4
5  CREATE TABLE Professeur (
6  Numss VARCHAR(13) PRIMARY KEY,
7  Nom CHAR(20) NOT NULL,
8  NomUniv CHAR(50) REFERENCES Univ(Nom));
9
10 CREATE TABLE  Doctorant (
11 Numss VARCHAR(13) PRIMARY KEY,
12 Nom CHAR(20) NOT NULL,
13 NomUniv CHAR(50) REFERENCES Univ(Nom),
14 EncadrePar VARCHAR(13) REFERENCES Professeur(Numss));
```

B. Exercices

1. À l'école II

[10 min]

Une école est doté d'une base de données pour gérer ses classes. On dispose des modèles UML et relationnel suivant décrivant cette base de données.




```
1  classe (#id:entier(1)-char(1),  type{normal|sport_étude|
   bilingue_anglais|bilingue_allemand})
```

Question

Montrer que ce schéma n'est pas en première forme normale, et proposer une correction du schéma relationnel.

2. Project manager

[20 min]

Soit la table suivante (Emp signifie *Employee*, Pro signifie *Project* et Man signifie

Manager).

Emp	EmpName	Pro	ProName	Man	ManName
E01	John	P1	Eco	M1	Becky
E02	Mary	P2	Admin	M2	Joe
E03	Mark	P2	Admin	M2	Joe
E04	Travis	P3	Educ	M1	Becky

Soit les dépendances fonctionnelles suivantes :

- Emp → EmpName
- Emp → Pro
- Pro → ProName
- Pro → Man
- Pro → ManName
- Man → ManName

Question 1

Montrer que ce modèle n'est pas en 3NF.

Question 2

Proposer un modèle équivalent en 3NF.

Question 3

Réécrivez les données en respectant ce nouveau schéma.

3. Objectifs II

[25 min]

Vous êtes le président de l'association "Objectifs", dont l'objet est d'aider ses étudiants membres à mener des projets dans le cadre de leurs études. Pour le moment cette association gère tous ces projets avec un logiciel de type tableur.

Numéro	Projet / Tâche	Spécialités	Chef de projet	Participant tâche	Dates	Partenaire	
1	La nuit du Picolo	Logistique	Paul Densmore ; Spécialiste Musique		25 décembre	1666 (Brasseur)	Bières offertes
1,1	Gestion			Barbara Krieger ; Spécialiste Sport			
2	Escalade de l'Everest	Voyages	Paul Manzarek		Intersemestre	Pentathlon 1666 (Brasseur)	Fourniture de matériel Publicité
3	Tournoi de Volley-Ball	Sport	Paul Manzarek		15/5 - 23/5		
3.1	Recrutement des équipes			Barbara Krieger ; Spécialiste Sport			
3.2	Gestion			Paul Densmore ; Spécialiste Musique			

Exemple de fichier de gestion des projets de l'association Objectifs

La concision et la clarté de la rédaction, ainsi que la bonne mobilisation des concepts et de la terminologie du domaine des bases de données seront intégrées à l'évaluation.

Question 1

On considérera ce tableau comme une relation, et on admettra que cette extension est suffisante pour l'interprétation des données :

1. **Démontrer** que cette relation ne peut admettre qu'une seule clé.
2. **Démontrer** que la 1NF n'est pas respectée à plusieurs reprises (on peut trouver jusque six causes empêchant la 1NF, proposez-en un maximum).

Question 2

On propose une première décomposition pour que le modèle soit en 1NF.

```
1  Projet (#Num, Nom, Debut, Fin, Specialite, CDP-Prenom, CDP-Nom, CDP-
   Specialite)
2  Tache (#NumProjet=>Projet, #NumTache, Nom, Participant-Prenom,
   Participant-Nom, Participant-Specialite)
3  Partenaire (#Nom, Description)
4  Partenariat (#Partenaire=>Partenaire, #Projet=>Projet, role)
```

1. Montrer que ce modèle n'est pas en 3NF et lister l'ensemble des DFE responsables.
2. Proposer une décomposition en 3NF.

4. Jeu de construction

[40 min]

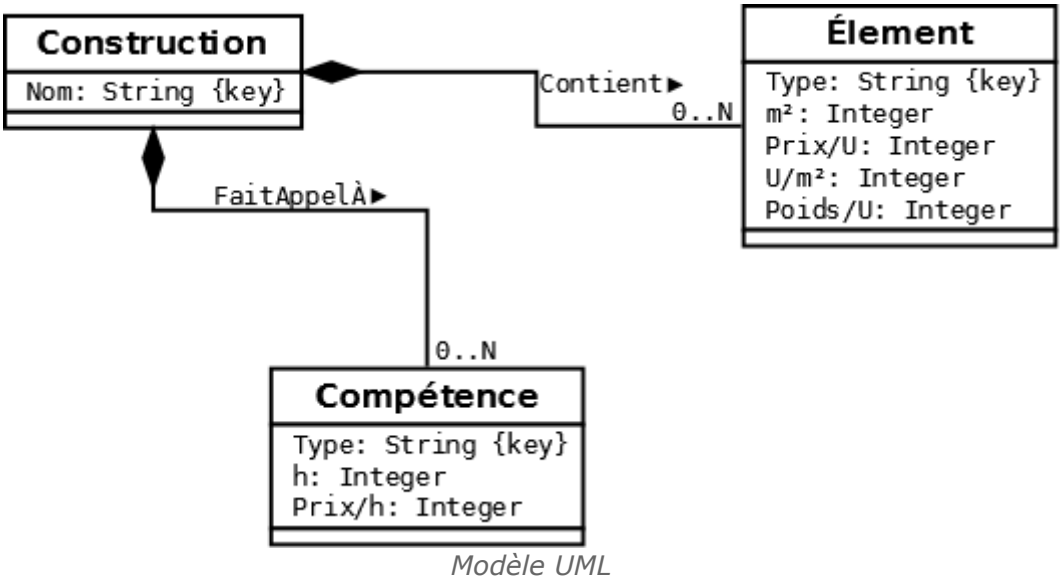
Soit le diagramme UML ci-après décrivant des **constructions** composées d'**éléments** et faisant appel à des **compétences** pour leur montage.

Par exemple une "Maison de type I" contiendra :

- 100 m² de "Brique traditionnelle" (Type de l'élément), pour un prix de 2 euros par unité (Prix/U), 12 briques par m² (U/m²) et 100g / brique (Poids/U)
- 125 m² de "Tuile plate" (Type de l'élément), pour un prix de 4 euros par unité (Prix/U), 24 tuile par m² (U/m²) et 75g / tuile (Poids/U)

et fera appel à :

- 500h de travail de "Maçon" (Type de compétence), pour un tarif horaire de 20€/h
- 425h de travail de "Couvreur" (Type de compétence), pour un tarif horaire de 30€/h
- 75h de "Chef de chantier" (Type de compétence), pour un tarif horaire de 45€/h



Modèle UML

Question 1

Effectuer le passage au relationnel de ce modèle.

Question 2

Sachant que deux mêmes types d'élément ont **toujours** les mêmes caractéristiques (Prix/U, U/m², Poids/U) et que deux mêmes types de compétence ont **toujours** les mêmes prix (Prix/h), établir la fermeture transitive des DFE et la forme normale de ce modèle (justifier).

Donner un exemple de données redondantes.

Indice :

> *Deux types de maison ont évidemment des quantités d'éléments (m²) et de compétences (h) différentes.*

Question 3

Normaliser le modèle en 3NF (justifier, et montrer que la transformation est sans perte).

Question 4

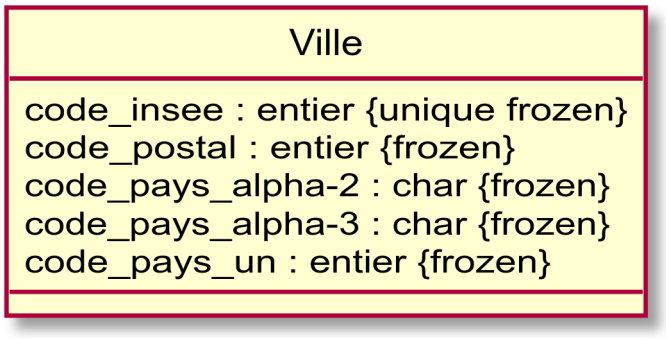
En repartant de la forme normalisée en 3NF, rétro-concevoir un modèle UML qui aurait permis d'aboutir directement à ce schéma non redondant.

5. Codes normalisés

[15 min]

On considère la classe Ville ci-après et on renomme les attributs comme suit :

- code_insee : ci
- code_postal : cp
- code_pays_alpha-2 : a2
- code_pays_alpha-3 : a3
- code_pays_un : un



On pose les DF suivantes :

- ci → cp, a2, a3, un
- a2 → a3, un
- a3 → a2, un
- un → a2, a3

Question 1

Exprimez la fermeture transitive et trouvez les clés.

Question 2

Le modèle est-il en 3NF ? Justifiez et si nécessaire, normalisez.

6. ÀDeuxMains

[30 min]

Une entreprise de maintenance de matériel industriel

La société ÀDeuxMains est en charge de la maintenance de machines industrielles. La société gère plusieurs agences ; chaque agence est affectée à exactement une région dans le monde.

Agences

On dispose d'un fichier CSV `region.csv` correspondant au schéma suivant :

```
1  Agence (numéro_agence:integer, nom_région:varchar,
   population_région:integer, surface_région:integer,
   densité_région:real, nom_pays:varchar, pib_pays:integer)
```

On dispose des informations complémentaires suivantes :

- numéro_agence est unique et non null
- nom_région est unique et non null
- densité_région = population_région / surface_région
- nom_pays détermine pib_pays

On ne fera aucune hypothèse de dépendance fonctionnelle complémentaire.

Question 1

Montrez que ce modèle n'est pas en 3NF (rappelez les clés candidates et énoncez la liste des DFE sous la forme d'une F+). Proposez une solution de normalisation **en relationnel** (sans perte).

Proposez un modèle UML correspondant à la version normalisée.

Machines

On dispose de la liste des types de machines gérés par ÀDeuxMains dans un fichier de tableur *types.ods*.

#machine	spécialités
Compressor 08	Électricité / Mécanique
Aérienne	Électricité
Mxte	Électricité / Plomberie
Aquatique	Électricité / Plomberie / Mécanique
Full Fuel	Plomberie
Naturéo Carbone	Plomberie
Flash	Électricité

On sait que les noms des machines sont des clés.

Question 2

Montrez que ce modèle n'est pas en 3NF. Proposez une solution de normalisation **en relationnel** (sans perte).

Proposez un modèle UML correspondant à la version normalisée.

Index